OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Applicability of Quantum Computing on Database Query Optimization

A Thesis

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Science (M.Sc.)

At

Regensburg University of Applied Sciences

Student Name: Manuel Schönberger
Student Number: 3248320

Primary Supervising Professor: Prof. Dr. Wolfgang Mauerer
Secondary Supervising Professor: Prof. Dr. Stefanie Scherzinger

Submission Date: 22.12.2021

Erklärung zur Masterarbeit von

Name: Schönberger

Vorname: Manuel

Studiengang: Master Informatik

1. Mir ist bekannt, dass dieses Exemplar der Masterarbeit als Prüfungsleistung in das Eigentum der Ostbayerischen Technischen Hochschule Regensburg übergeht.
2. Ich erkläre hiermit, dass ich diese Masterarbeit selbständig verfasst, noch nicht anderweitig für Prüfungszwecke vorgelegt, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate als solche gekennzeichnet habe.

Regensburg, den 22.12.2021

Manuel Schönberger

Abstract

This thesis analyzes the applicability of quantum computing on database query optimization. More specifically, both the multi query optimization (MQO) problem as well as the join ordering problem are investigated to this end. An approach for solving MQO on a D-Wave quantum annealing machine has previously been proposed and experimentally evaluated. For the join ordering problem, this work presents a two-step transformation in which the problem is first reformulated as a binary integer linear programming problem, based on an existing method. The transformed problem can then be formulated as a quadratic unconstrained binary optimization (QUBO) problem, which is a suitable formulation for current quantum systems.

This work further investigates the applicability of the existing approach for MQO via simulations with respect to state-of-the-art gate-based quantum systems, specifically IBM-Q machines, in comparison to the existing results for quantum annealing. Similar simulations for both gate-based and quantum annealing systems are conducted to evaluate the proposed approach for solving join ordering problems. Two variational hybrid quantum-classical algorithms, the variational quantum eigensolver (VQE) and the quantum approximate optimization algorithm (QAOA), are simulated for current IBM-Q machines. Simulation results show that QAOA can in nearly all cases be more reliably executed than VQE with respect to the limited coherence time of current systems. Moreover, for both problems, existing D-Wave quantum annealing systems can solve significantly larger problems than current gate-based IBM-Q systems.

However, these problem sizes are still limited compared to the problem dimensions solvable via classical approaches. Still, considering the rapid evolution of quantum systems, the prospect of using sufficiently mature quantum devices, which may become available in the near future, for practical problem dimensions is promising and motivates further research to identify further suitable database optimization problems.

Contents

List of Figures

List of Tables

List of Abbreviations

BILP binary integer linear programming
MILP mixed integer linear programming
MQO multi query optimization
NISQ noisy intermediate-scale quantum
PPQ plans per query
QAOA quantum approximate optimization algorithm
QPU quantum processing units
QUBO quadratic unconstrained binary optimization
SDK software development kit
VQE variational quantum eigensolver

1 Introduction

Quantum computing seeks to use properties and phenomena of quantum mechanics in order to achieve computational speedups. It has been shown that certain problems can indeed be solved faster with quantum algorithms in comparison to classical computing. Two famous quantum algorithms that demonstrate the advantage of quantum computing over classical computing for certain problems are Grover's algorithm and Shor's algorithm [1], [2]. The former can be used to find an element in an unsorted database with time complexity $\mathcal{O}(\sqrt{n})$ and thus has a quadratic speedup over classical algorithms [1]. The latter efficiently solves the problem of prime factorization and finding discrete logarithms [2].

The first quantum computing machines have already been built. For instance, IBM offers access to its gate-based quantum systems as a cloud service [3]. However, the currently available devices are classified as so-called noisy intermediate-scale quantum (NISQ) systems, which are prone to different kinds of errors that limit their capabilities [4]. For instance, errors can occur during the execution of quantum gates or due to the decoherence of quantum states caused by interactions with the environment of the system. Moreover, current machines generally only feature a limited number of qubits, which puts strict restrictions on the size of problems that can be solved on them.

However, certain quantum computing approaches are promising even for quantum systems in the NISQ era. For instance, variational hybrid quantum-classical algorithms such as the variational quantum eigensolver (VQE) and the quantum approximate optimization algorithm (QAOA) are particularly suitable for near-term gate-based quantum machines [4]–[7]. By performing only a limited number of steps on a quantum computer and the remaining ones on classical machines, the task of leveraging the benefits of quantum computing and yet using current quantum systems becomes more feasible.

Moreover, the company D-Wave has built several machines for adiabatic quantum computing. More specifically, these machines perform quantum annealing, which can be understood as adiabatic quantum computing on noisy devices [4]. The newest available D-Wave machine at the time of writing is the D-Wave Advantage system [8]. These quantum annealing systems can be used for solving a specific type of optimization problem.

Quantum annealing machines feature a significantly higher number of qubits than current gate-based devices. For instance, the D-Wave Advantage system offers over 5,000 qubits, whereas the largest available IBM-Q system in terms of qubit numbers at the time of writing features 65 qubits. However, as a result of embedding the optimization problems onto the topology of a D-Wave system, several physical qubits connected in a chain are typically needed to represent one logical qubit, making the number of effectively available logical qubits significantly lower.

Still, various different optimization problems have already been solved using the D-Wave quantum annealing machines. Among these are also problems in database query optimization. More

specifically, the application of quantum annealing on multi query optimization (MQO) problems was previously analyzed in [9]. Given the limitations of current devices, only smaller-sized problem instances could be solved on the quantum annealing system. Despite that, the authors of [9] were able to identify certain MQO problem classes for which the used quantum annealing system was superior in comparison to other approaches based on classical machines.

Since several new gate-based quantum systems have been made available by IBM in recent years, similar results might be achievable on gate-based systems through the use of the above-mentioned variational hybrid algorithms. As such, this work aims to analyze the applicability of the hybrid quantum-classical algorithms on the MQO problem using current IBM-Q systems in comparison to the results for the quantum annealing approach presented in [9].

Another highly relevant problem in the field of query optimization is the join ordering problem, which features a vast search space [10]. Since such problems are particularly challenging to handle with classical approaches, the prospect of using quantum computing as a promising alternative is particularly attractive [11]. Similar to the approach shown in [9] for MQO, it might be possible to leverage quantum computing by bringing the problems into a form suitable for current gate-based quantum devices and quantum annealers. As such, the second goal of this work is to find such a reformulation method and to analyze it with respect to the applicability of both currently available gate-based systems and quantum annealing machines.

In summary, this work aims to answer the following research questions:

1. How does the applicability of variational hybrid quantum-classical algorithms on MQO problems, using state-of-the-art gate-based quantum systems, compare to previous results for a quantum annealing machine?
2. How can the join ordering problem be reformulated for quantum computing and how applicable is this approach for current quantum systems with respect to the solvable problem dimensions?

Structure. The remainder of this work is structured as follows: Chapter 2 describes how this work relates to prior research. Chapter 3 gives an overview on quantum computing and further describes the state of currently available quantum systems. Chapter 4 gives an overview on the two above-mentioned query optimization problems. Chapters 5 and 6 respectively describe how MQO and the join ordering problem can be reformulated into a form suitable for quantum computing and present simulation results, which are discussed in Chapter 7. Finally, Chapter 8 concludes this work.

2 Related Work

MQO has been a topic of extensive prior research in database query optimization. In [12], a detailed explanation on MQO is given and different types of algorithms, some of which only consider locally optimal plans whereas others are heuristic in nature, are presented. Since the search space for MQO is of size $\mathcal{O}(n^n)$, exhaustive algorithms only provide limited use and most of the existing approaches for finding optimal solutions for the problem are heuristic instead [13]. This includes an algorithm presented in [13], which contains a reformulation of MQO as an unconstrained, normalized submodular maximization problem.

An approach for MQO which relies on a genetic algorithm has been proposed in [14] and has been shown to produce good results for larger MQO problems. Finally, an approach which allows MQO to be solved using a quantum annealer after reformulating the problem as a quadratic unconstrained binary optimization (QUBO) problem is presented in [9]. Despite the limited dimension of MQO problems solvable on the quantum annealer, the authors of [9] identified classes of MQO problems for which the quantum annealing device was superior in comparison to classical machines.

The join ordering problem has also been intensively researched. In [10], a detailed explanation of the join ordering problem is given and it is moreover described how a join ordering problem may be classified, e.g., by taking into account which type of cost function and join tree is considered. Different kinds of algorithms, namely heuristic, randomized and genetic ones, have been studied [15]. The latter two were found to produce better results than heuristic algorithms, albeit taking longer to execute. Moreover, a method for reformulating the join ordering problem as an mixed integer linear programming (MILP) problem in order to make use of mature MILP solvers has been presented in [16]. This MILP approach has been found to be applicable for join ordering problems that were too large for solvers that rely on other approaches.

This work contributes to existing research in the field of query optimization in several ways. Firstly, it builds on [9] and, based on the MQO reformulation method into a QUBO problem, examines the applicability of hybrid quantum-classical algorithms with respect to the properties of current gate-based IBM-Q quantum systems. It moreover compares the results for a state-of-the-art IBM-Q system to the existing ones for the D-Wave quantum annealing system used in [9] with regards to the problem dimensions that are possible to solve on the respective systems.

Another contribution of this work is the identification of an approach that allows the join ordering problem to be solved on current quantum systems. In order to achieve this, this work builds on [16] and makes use of the reformulation method for join ordering problems into MILP problems in order to ultimately bring the problem into a form suitable for current quantum systems. However, this work only considers a more basic MILP formulation in order to lower the qubit requirements, whereas [16] also describes various extensions which make the formulation more sophisticated.

Finally, this work contributes to existing research for the join ordering problem by evaluating the reformulation approach for both quantum annealing as well as gate-based quantum systems in terms of the solvable problem dimensions. The evaluation results are moreover compared to the results for a classical MILP solver described in [16]. Unlike other works such as [9], the results for this work are gained exclusively via simulations and not via the use of real quantum systems. Moreover, only the size of the solvable problem dimensions is taken into account for the results of this work, whereas other criteria such as the quality of the solutions are not investigated.

3 Background on Quantum Computing

This chapter will give an overview on quantum computing. First, Section 3.1 will explain the difference between classical bits and qubits and describe the concept of superpositions. Next, Section 3.2 will give an overview on quantum gates and circuits, which serve as the basis for gate-based quantum computing. Section 3.3 will then briefly introduce the concept of Hamiltonians, which are used for both the variational hybrid quantum-classical algorithms explained in Section 3.4 as well as for adiabatic quantum computing described in Section 3.5. Finally, Section 3.6 will give an overview on the state of current quantum systems, as well as their limitations.

3.1 Qubits and Superpositions

Similar to classical bits, which can be in either the state 0 or 1, a qubit can also occupy different states, such as $|0\rangle$ and $|1\rangle$ [17]. However, in contrast to classical bits, which can only be in one of two states at a time, qubits can be in a so-called *superposition* of states [18]. This superposition can be expressed as a linear combination of states:

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

where the states $|0\rangle$ and $|1\rangle$ are the computational basis states and the coefficients α and β are complex numbers [17]. These coefficients are also referred to as *amplitudes*. However, in accordance to quantum mechanics, such a superposition of states only persists as long as the qubit is not observed. After measuring the state of $|\psi\rangle$, the superposition will collapse into either one of the computational basis states. The probability of the superposition collapsing into a specific basis state is given by the square of its corresponding amplitude, in the above case $|\alpha|^2$ for $|0\rangle$ or $|\beta|^2$ for $|1\rangle$ [17].

The notation used for the states which was applied above is called the *Dirac notation*, which is explained in [18]. A state such as $|\psi\rangle$ is called a *ket*, which is in this case labeled by ψ, and is moreover a linear combination of vectors so that $|\psi\rangle = a_1\,|s1\rangle + a_2\,|s2\rangle + ... + a_n\,|s_n\rangle$, where a_i with $1 \leq i \leq n$ is the amplitude for the state $|si\rangle$. Since all squared amplitudes need to add up to 100%, $|\psi\rangle$ is normalized to length 1 [17]. A set of vectors B which can be used to uniquely express each state in a quantum system as a linear combination makes up a basis for this system.

Furthermore, $\langle\psi|$ is called a *bra* with the label ψ and is the conjugate transpose of $|\psi\rangle$:

$$|\psi\rangle = \begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \qquad\qquad \langle\psi| = (\overline{a_1}, \overline{a_2}, ..., \overline{a_n}). \tag{2}$$

It can be seen that $|\psi\rangle$ is a column vector whereas $\langle\psi|$ is a row vector. The inner product of two state vectors $|v1\rangle$ and $|v2\rangle$ is then given by $\langle v1|v2\rangle$. Due to the normalization condition mentioned above, $\langle\psi|\psi\rangle = 1$ must hold for any quantum state $|\psi\rangle$. However, if $\langle v1|v2\rangle = 0$, then the vectors $|v1\rangle$ and $|v2\rangle$ are called *orthogonal*. Moreover, a set of vectors is called *orthonormal* if each pair of vectors in the set is orthogonal and if for each vector $|v\rangle$ in the set, $\langle v|v\rangle = 1$ holds. The orthonormal condition plays a big role in quantum computing, since a base for a quantum system typically needs to have this property. One possible set of state vectors to express the basis states $|0\rangle$ and $|1\rangle$ which fulfills the orthonormal condition is given by $\begin{pmatrix}1\\0\end{pmatrix}$ for $|0\rangle$ and $\begin{pmatrix}0\\1\end{pmatrix}$ for $|1\rangle$. Another orthonormal basis which is important in quantum computing is given by the states $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ and $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$. It can be seen that these basis states correspond to a balanced superposition, which collapses to either $|0\rangle$ or $|1\rangle$ with equal probability upon measurement.

As described in [17] and with consideration of the equation $\sin^2 x + \cos^2 x = 1$, Equation 1 can also be expressed by

$$|\psi\rangle = e^{i\gamma}\left(\cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle\right), \tag{3}$$

where θ, φ, $\gamma \in \mathbb{R}$. The factor $e^{i\gamma}$ is a global phase, which means it has no observable effect and can effectively be ignored for the following considerations. Parameterized by θ and φ, any single-qubit quantum state can be visualized by the so-called Bloch sphere.

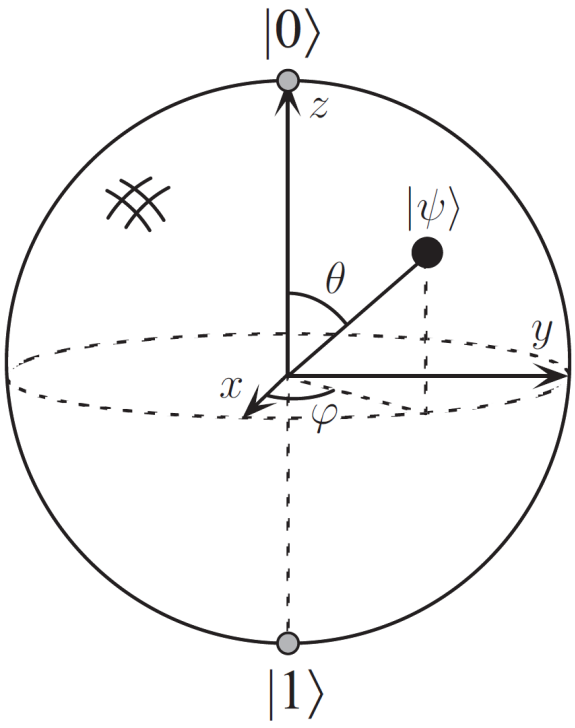

Figure 1: Bloch sphere for visualizing single qubit states (source: [17]).

Figure 1 depicts the Bloch sphere, which is a three-dimensional space where the computational basis states $|0\rangle$ and $|1\rangle$ are positioned on opposing polar ends. As explained in [18], the surface of the Bloch sphere contains the entire state space of a single-qubit system. For instance, the state $|+\rangle$ is represented by the surface point which is touched by the positive x-axis.

While the Bloch sphere is useful for visualizing the possible states of a one-qubit quantum system, it cannot be used on systems with more than one qubit. However, multi-qubit systems are much more relevant than their single-qubit counterparts, since this is where the computational advantage of quantum computing comes into play. As described in [17], for a quantum system with two qubits, the superposition for a state is given by

$$|\psi\rangle = \alpha_{00}\,|00\rangle + \alpha_{01}\,|01\rangle + \alpha_{10}\,|10\rangle + \alpha_{11}\,|11\rangle\,, \tag{4}$$

where $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$ are the four computational basis states and α_{ij} with $i, j \in \{0, 1\}$ are their respective probability amplitudes. Like before, the normalization condition requires the squared amplitudes to add up to 1. In general, a single state of an n-qubit quantum system is described by 2^n amplitudes. As such, the information stored by such a quantum state vastly exceeds the information contained in a classical state. Quantum computation seeks to make use of the large amount of information that can be stored in the states of quantum systems.

Quantum states of multi-qubit systems can also be described as vectors in a vector space of 2^n dimensions. The vector space of a two-qubit system, which has 4 dimensions, can be considered a combination of the vector spaces of two single-qubit systems, each of them having a dimension of 2. As explained in [17], in general the combination of vector spaces V and W with m and n respective dimensions can be expressed as $V \otimes W$. The $\otimes$-operation is referred to as the *tensor product*.

The resulting vector space has mn dimensions and its orthonormal basis is given by $|i\rangle \otimes |j\rangle$ where $|i\rangle$ and $|j\rangle$ are the bases of W and V respectively. Moreover, the states of the new vector space are linear combinations of $|v\rangle \otimes |w\rangle$ where $|v\rangle$ and $|w\rangle$ are states of V and W. In this work, a tensor product $|v_1\rangle \otimes |v_2\rangle \otimes ... \otimes |v_n\rangle$ will henceforth be abbreviated by $|v_1 v_2 ... v_n\rangle$, which is a commonly used abbreviation. For matrices and vectors, this operation can be represented by the Kronecker product as described in [17]. For instance, using the state vector for $|0\rangle$ for both operands, the operation works as follows:

$$|0\rangle \otimes |0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \otimes \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \cdot 1 \\ 1 \cdot 0 \\ 0 \cdot 1 \\ 0 \cdot 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{5}$$

It can be seen that the elements of the first operand are pairwise multiplied with the second operand to obtain the result.

Finally, quantum computing can utilize a peculiar phenomenon that is exclusive to quantum mechanics. As explained in [18], the state of a classical state space can be described through all individual states of all components that make up the space. In contrast, the state of a quantum system can often times not be characterized by considering the states of all separate components. This is due to possible correlations between several states, which are then referred to as *entangled* states. The phenomenon of entanglement serves as the basis for many quantum computing approaches.

3.2 Quantum Gates and Circuits

This section will give an overview on quantum computation based on the quantum states explained in the previous section. Specifically, it will describe how these quantum states are transformed into other states. As explained in [18], a quantum state is transformed by rotating its state vector in the complex vector space. The transformation of a vector can obviously be done by applying a matrix. Moreover, there are restrictions on the properties a matrix must have in order to perform valid quantum state transformations. Specifically, valid transformations require the corresponding matrix to be *unitary*.

For a matrix U to be unitary, the following condition must hold as described in [17]:

$$U^\dagger U = I, \tag{6}$$

where I is the identity matrix and $U^\dagger$ is the transpose conjugate of U. The transformation of a state vector $|\psi\rangle$ using a unitary matrix U preserves the inner product of $|\psi\rangle$ [18]. As such, the transformed state vector $|\psi^*\rangle$ also fulfills the normalization condition explained in the previous section if $|\psi\rangle$ is a valid quantum state vector.

Practically, a quantum computer performs such quantum state transformations through gates, which are parts of circuits and connected via wires [17]. More specifically, gates represent unitary matrices. The final operations performed by a quantum circuit are typically measurements on the qubit states which have been prepared by applying gates. More details about how such measurements are done will be explained below. In the following, an overview on important single qubit gates and multiple qubit gates will be given.

Some of the most relevant gates which act on a single qubit are the so called *Pauli* gates [17]. They are given by

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \qquad Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}. \tag{7}$$

For instance, applying X on the computational basis states produces

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad X|1\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{8}$$

It can be seen that X performs a negation as it transforms $|0\rangle$ into $|1\rangle$ and vice versa. More generally, the first column of a single qubit matrix shows how the computational basis state $|0\rangle$ is transformed whereas the second column does so for $|1\rangle$. As such, it is obvious that Z leaves $|0\rangle$ unchanged while it transforms $|1\rangle$ to $-|1\rangle$. Sometimes, the identity matrix is also considered a Pauli matrix.

One property that makes the Pauli matrices special is that they are not only unitary, but also *Hermitian* matrices. As explained in [18], for a matrix A to be Hermitian, the following must hold:

$$A = A^{\dagger}. \tag{9}$$

As explained in [18], all eigenvalues of Hermitian operators are real. Moreover, Hermitian operators are sometimes also referred to as *observables* when they are used for measurements. More specifically, the eigenvectors of A are the possible states after the measurement. For instance, the Pauli Z operator has the eigenvectors $|0\rangle$ and $|1\rangle$ with the corresponding eigenvalues 1 and -1. As such, Z can be used for measurements in the computational basis and is thus a commonly used observable. Hermitian operators play an important role in the context of both the hybrid quantum-classical algorithms as well as adiabatic quantum computing which will be discussed in later sections.

Another important single qubit matrix is the Hadamard matrix H, which is also Hermitian:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{10}$$

The Hadamard operator transforms the computational basis states $|0\rangle$ and $|1\rangle$ into the balanced superposition states $|+\rangle$ and $|-\rangle$ respectively. As such, the Hadamard gate is often among the first gates used in a quantum circuit in order to create a superposition.

Furthermore, the *controlled-NOT* or *CNOT* gate is one of the most relevant gates acting on multiple qubits. As explained in [17], it is given by the following matrix:

$$CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{11}$$

The first qubit the *CNOT* gate acts on is referred to as the control qubit. The state of the control qubit determines whether or not the second qubit, also called the target qubit, will be transformed. If the control qubit is $|1\rangle$, the target qubit will be flipped. Otherwise, the state

of the target qubit remains unchanged. Together with the Hadamard gate, the *CNOT* gate is oftentimes used to create entangled states.

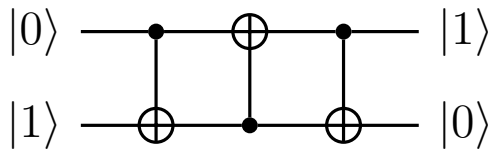

Figure 2: Circuit for swapping the states of two qubits.

Figure 2 depicts a quantum circuit, which, as explained in [17], has the useful effect of swapping the states of the two qubits. This is illustrated by the specific scenario depicted in Figure 2, where the two-qubit state $|01\rangle$ is transformed into the state $|10\rangle$. This is achieved by applying three *CNOT* gates, the first and third one using the first qubit as the control qubit whereas the second *CNOT* gate uses the second qubit as the control qubit.

In the context of practical quantum computing, in particular when considering current state-of-the-art gate-based quantum computers, this swap operation plays a crucial role. More specifically, it can be used to create indirect connections between physical qubits that are not direct neighbours. More details about the important role of the swap gate for state-of-the-art quantum systems will be discussed in a later section.

Finally, as explained in [17], similar to how in the classical case, some sets of gates or even single gates like the *NAND* gate can be used to create any other gate and thus allow for universal computing, there exists a set of quantum gates which can be used to create any quantum circuit. In general, a set of gates is universal for quantum computing if it can be used to approximate any unitary operator to arbitrary accuracy. The *CNOT* gate, alongside single qubit gates, can be utilized for this purpose. Regarding single qubit gates, it is possible to approximate any gate to arbitrary accuracy using only a limited number of gates. As such, universal quantum computation is possible with a limited set containing such single qubit gates as well as the *CNOT* gate.

3.3 Encoding Optimization Problems as Ising Hamiltonians

This section will briefly explain the optimization problem encoding that is used by all quantum computing approaches that will be discussed in the following sections. Typically, an optimization problem that is to be solved with quantum computing on currently available quantum systems is formulated as a *Hamiltonian*, which is a Hermitian operator describing the dynamics of a quantum system and which represents the energy of the system [17]. As explained in [6],

a Hamiltonian H can be decomposed and written as

$$H = \sum_{i,\alpha} h^i_\alpha \sigma^i_\alpha + \sum_{i,j,\alpha,\beta} h^{i,j}_{\alpha,\beta} \sigma^i_\alpha \sigma^j_\beta + ... \ , \tag{12}$$

where h is a real value and σ_α, σ_β are Pauli operators. The indices i, j denote the subspace which the operators act on. For this work, a more restricted type of Hamiltonian is considered, which is typically required for current quantum systems and where each operator acts on at most two qubits. Specifically, the *Ising* model is used for the formulation of optimization problems. As explained in [19], [20], the Ising model is given by

$$H(s_1, ..., s_N) = -\sum_{i<j} J_{ij} s_i s_j - \sum_{i=1}^{N} h_i s_i, \tag{13}$$

where $s_i \in \{-1, 1\}$ and J_{ij} as well as h_i are real numbers. The corresponding Hamiltonian for quantum computing is gained by replacing a spin s_i with Pauli operators similar to how it is done in Equation 12. Quantum computing approaches dealing with Hamiltonians typically seek to determine their minimum energy level. As such, an optimization problem needs to be formulated as a Hamiltonian in such a way that its ground state encodes an optimal solution.

Moreover, the Ising model is equivalent to the QUBO problem formulation [21]. More precisely, instead of spins, boolean variables are used for QUBO formulations, which makes the transformation from an Ising model to QUBO (and vice versa) straightforward as explained in [21]. As such, the remainder of this work will treat both formulations as interchangeable.

3.4 Variational Hybrid Quantum-Classical Algorithms

This section will give an overview on two algorithms for solving optimization problems based on quantum computing. More specifically, the algorithms are hybrid algorithms as they contain steps for both classical machines as well as quantum systems. First, Section 3.4.1 will explain the VQE algorithm. Next, the QAOA algorithm will be described in Section 3.4.2.

3.4.1 Variational Quantum Eigensolver

One algorithm utilizing quantum computing which can ultimately be used for solving optimization problems of a certain form is the VQE algorithm [6]. More specifically, in addition to the quantum computational parts, VQE also contains steps which can be performed on a classical computer. This hybrid approach is particularly suitable for current quantum computers, which are both limited in the problem dimensions they can handle and prone to errors. More details about these limitations will be discussed in a later section. By decomposing an algorithm into several parts and by only performing certain parts on a quantum computer, the algorithm becomes more applicable for current quantum systems.

As explained in [6], the goal of VQE is to determine the smallest eigenvalue of a Hamiltonian, which corresponds to finding its lowest energy value as explained in Section 3.3. One of the most well-known applications of VQE is the field of quantum chemistry: Here, VQE can be used to calculate the ground state energy of a molecule [6]. However, VQE can also be used for more general optimization problems that are encoded by Hamiltonians in such a way that their respective ground state corresponds to an optimal solution.

The foundation for VQE is the variational principle, which is explained in [5]. First, the expectation value for an operator O and a state $|\psi\rangle$ is given by

$$\langle O \rangle_{|\psi\rangle} = \frac{\langle \psi | O | \psi \rangle}{\langle \psi | \psi \rangle}. \tag{14}$$

The denominator containing the inner product $\langle \psi | \psi \rangle$ can be ignored, as it always equals 1 since only normalized quantum states are considered in this work. The variational principle is given as follows:

$$\langle H \rangle_{|\psi(\theta)\rangle} \equiv \langle H \rangle (\theta) = \langle \psi(\theta) | H | \psi(\theta) \rangle \geq \lambda_{\min}, \tag{15}$$

where H is a Hamiltonian, $|\psi(\theta)\rangle$ is a quantum state that has been prepared depending on the parameter θ and $\lambda_{\min}$ is the smallest eigenvalue of H. The expectation value is thus an upper bound for the smallest eigenvalue that VQE seeks to determine. The smallest eigenvalue can be determined by fine-tuning the parameter θ until the expectation value with regards to the state $|\psi(\theta)\rangle$ is minimized. This is the general idea behind the VQE algorithm.

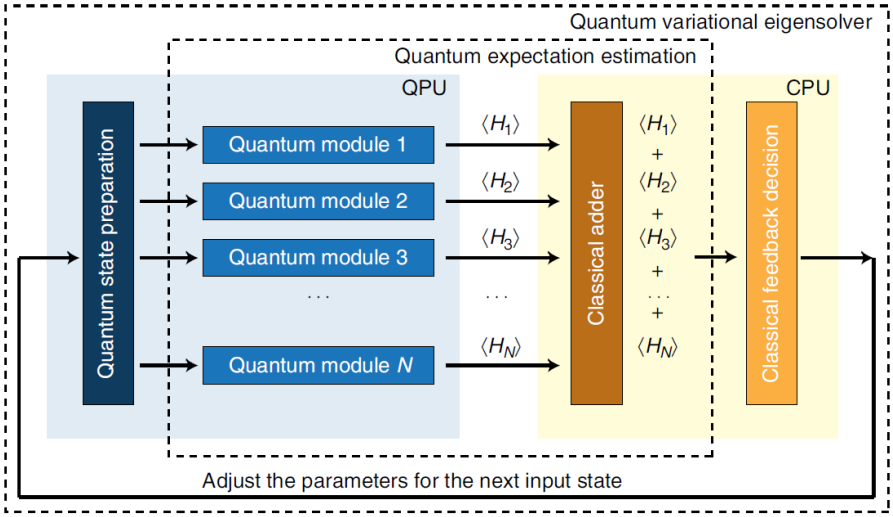

Figure 3: The steps of the variational quantum eigensolver (source: [6]).

Figure 3 shows the steps of the VQE algorithm. In the first step, the quantum state is prepared on the quantum computer with respect to the parameter θ. Next, the expectation values for the individual terms of the Hamiltonian are computed on a quantum system via repeated sampling. Based on the individual results $\langle H_1 \rangle, ..., \langle H_N \rangle$, the overall expectation value for H is determined

on a classical machine. Finally, depending on this expectation value, θ is adjusted and once again used for the state preparation for the next iteration of the algorithm. This process is repeated until the upper bound for the smallest eigenvalue converges.

3.4.2 Quantum Approximate Optimization Algorithm

Another hybrid algorithm similar to VQE that is able to utilize quantum computing in addition to performing steps on a classical computer is the QAOA algorithm [7]. Much like the former, QAOA is a variational algorithm, since it prepares a quantum state depending on two parameters which are tuned in between iterations. However, the state preparation process for QAOA is more sophisticated when compared to VQE.

The QAOA algorithm seeks to determine optimal solutions for combinatorial optimization problems. In contrast to VQE, where the Hamiltonian encoding the optimization problem is merely used for measurements, the concrete optimization problem already plays a role for the preparation of the quantum state when using the QAOA algorithm. Similar to VQE, the optimization problem is formulated as a Hamiltonian such that the ground state represents an optimal solution.

As explained in [7], there are two kinds of unitary operators which are used for preparing the quantum state. The first one depends on the specific optimization problem and is given by

$$U(C, \gamma) = e^{-i\gamma C} = \prod_{\alpha=1}^{m} e^{-i\gamma C_\alpha}, \tag{16}$$

where C is the Hamiltonian encoding the problem and the angle γ is one of the parameters that are tuned in between iterations. C is a diagonal matrix and so its eigenvectors are the computational basis states. Moreover, their respective eigenvalues correspond to the objective function outcomes of the optimization problem. As described in Section 3.3, a Hamiltonian can be decomposed into separate terms that contain Pauli operators typically acting on a limited number of qubits. C_α denotes one of the m individual terms. In the case of this work, which considers only Ising Hamiltonians, a term depends on at most two qubits.

The Hamiltonian which is needed for the second unitary operator is typically referred to as the mixer Hamiltonian and is given by

$$B = \sum_{j=1}^{n} \sigma_j^x, \tag{17}$$

where n is the number of qubits and σ_j^x is a Pauli X gate. The second unitary operator is then given by

$$U(B, \beta) = e^{-i\beta B} = \prod_{j=1}^{n} e^{-i\beta\sigma_j^x}, \tag{18}$$

where β is the second parameter which is adjusted after each iteration of the algorithm.

The initial state $|s\rangle$ which is to be prepared by the algorithm is simply given by the uniform superposition over the computational basis states $|z\rangle$:

$$|s\rangle = \frac{1}{\sqrt{2^n}} \sum_z |z\rangle \,. \tag{19}$$

Based on the two described operators and the initial quantum state, the prepared quantum state $|\gamma, \beta\rangle$ is given by

$$|\gamma, \beta\rangle = U(B, \beta)U(C, \gamma)...U(B, \beta)U(C, \gamma)\,|s\rangle \,. \tag{20}$$

It can be seen that the initial state is prepared by repeatedly alternating between applying the problem operator and the mixer operator. The number of repetitions p can be considered another parameter of the algorithm. As explained in [7], an upper bound for the circuit depth is given by $mp + p$. As such, it depends on the number of terms which make up the problem Hamiltonian. More specifically, this means a high number of two-qubit interactions in the case of Ising Hamiltonians results in larger circuit depths and thus has a negative impact on the applicability of the QAOA algorithm on current quantum systems, which will be discussed in more detail in later chapters. In contrast, the number of quadratic terms does not impact the circuit depth for the state preparation of the VQE algorithm.

The structure of the overall algorithm is similar to VQE. More specifically, the goal of the QAOA algorithm is to determine the parameters which produce a quantum state for which the expectation value $F_p(\gamma, \beta)$, which is given by

$$F_p(\gamma, \beta) = \langle \gamma, \beta | C | \gamma, \beta \rangle \,, \tag{21}$$

is maximized or minimized depending on the specific problem. Once an expectation value for the current quantum state $|\gamma, \beta\rangle$ has been determined, it is used for finding better parameters which are to be used for the next iteration. This process is performed on a classical computer. The algorithm terminates once the expectation value converges.

Furthermore, as shown in [7], let M_p be the maximum expectation value over the two parameters for p operator repetitions, then

$$\lim_{p->\infty} M_p = max_z C(z), \tag{22}$$

where z is the resulting bit string after measuring the prepared state in the computational basis. However, due to the strict limitations of current quantum systems, which will be discussed in later sections, it is likely that the resulting larger circuit depths caused by an increased number of operator repetitions negatively impact the quality of the algorithm execution.

3.5 Adiabatic Quantum Computing

Another quantum computing approach for solving optimization problems is given by adiabatic quantum computing [22], which provides an alternative to the circuit-based computational model discussed in earlier sections. It was shown in [23] that adiabatic quantum computation is equivalent to standard circuit-based quantum computation. Moreover, a term that is closely related to adiabatic quantum computing is quantum annealing.

While the two terms generally refer to the same method, they can be distinguished with regards to the conditions at which the algorithms are executed. Adiabatic quantum computing assumes a coherent adiabatic ground state evolution at zero temperature and thus ideal conditions, whereas quantum annealing does not [24]. The latter is thus more practically feasible and quantum systems on which this method can be performed have already been built.

The remainder of this section will give an overview on adiabatic quantum computing as explained in [22]. Once again, the method is centered around a problem Hamiltonian H_P which encodes the optimization problem so that its unknown ground state corresponds to an optimal solution. In addition, the algorithm includes the Hamiltonian H_B, which is constructed in such a way that its ground state is easy to determine. Then, a Hamiltonian representing an interpolation between H_B and H_P is given by

$$H(t) = (1 - t/T)H_B + (t/T)H_P. \tag{23}$$

The idea behind adiabatic quantum computation is to evolve the initial state at time step $t = 0$, which is the known ground state of $H(0) = H_B$, in a way that preserves the ground state during the evolution, which has an overall duration of T. If successful, the state at time step T will correspond to the ground state of $H(T) = H_P$, which represents an optimal solution to the optimization problem.

The ground state is preserved during the evolution process if certain conditions are met according to the adiabatic theorem. More specifically, the gap between the lowest energy level and the first excited energy level needs to be strictly greater than zero at all times. Then, the evolved state at $t = T$ will be in the ground state if the evolution process is done slowly enough by choosing a sufficiently high value for T. As such, the efficiency of the method depends on T.

As further described in [22], the following must hold for T:

$$T \gg \frac{\mathcal{E}}{g_{\min}^2}, \tag{24}$$

where $g_{\min}$ is the minimum energy gap between the ground state and the first excited state. Further explanation of the term $\mathcal{E}$ can be omitted and is not relevant for this work, since, as explained in [22], $\mathcal{E}$ is typically not very big and as a result, T depends on $g_{\min}^2$. This relation

should be taken into account when formulating the problem Hamiltonian, since an unsuitable formulation can lead to a small minimum energy gap and thus make the method inefficient.

3.6 State of Current Quantum Systems

This section will give an overview on currently available quantum systems and their properties. As already mentioned in Chapter 1, the current quantum machines are classified as NISQ systems, which means the systems have merely up to a few hundred qubits and are further impacted by errors causing noise [4]. Furthermore, these systems do not utilize quantum error correction, which could be used to mitigate these errors. Section 3.6.1 will give an overview on different kinds of errors which impact current gate-based quantum systems. Next, Section 3.6.2 will describe current quantum annealing systems.

3.6.1 Gate-Based Quantum Computing

Without quantum error correction, various different errors impose further limitations on the practicability of current gate-based quantum systems in the NISQ era in addition to their limited number of qubits. An overview on different kinds of errors is given in [4]. For one, errors may occur during the execution of a quantum gate, thus leading to noise during the circuit execution. The larger the number of gates in the quantum circuit, the more the execution is impacted by such gate errors. In [4], a circuit depth of around 1,000 is mentioned as the depth beyond which the execution of quantum circuits becomes impractical due to noise resulting from gate errors.

Another kind of error that needs to be considered is given by readout errors which may occur when conducting measurements. Furthermore, as explained in [25], errors related to crosstalk that arise due to the parallel execution of operations also need to be considered. However, such a parallel execution is necessary, since a serialized one would lead to large circuit depths and thus prolong the circuit execution time.

Since current quantum systems only have very limited coherence times, long circuit execution times are troublesome as they increase the likelihood of decoherence errors. Decoherence errors mean the loss of information about the state of the quantum system to the environment [18]. In order to avoid such errors, the circuit should be executed within the coherence time of the system. However, as described in [25], coherence decays exponentially. Thus, it is very important to reduce the execution time of a quantum circuit by keeping its depth as small as possible. However, due to the qubit topology of current quantum systems, achieving this goal is typically not a trivial task.

More specifically, not every qubit is connected with every other qubit on the qubit topology graph of current IBM-Q systems. Instead, the qubit topology for current quantum systems is typically sparse as illustrated in Figure 4, which shows the qubit topology for the IBM-Q Mumbai system containing 27 qubits. Without further measures, it is therefore not possible to

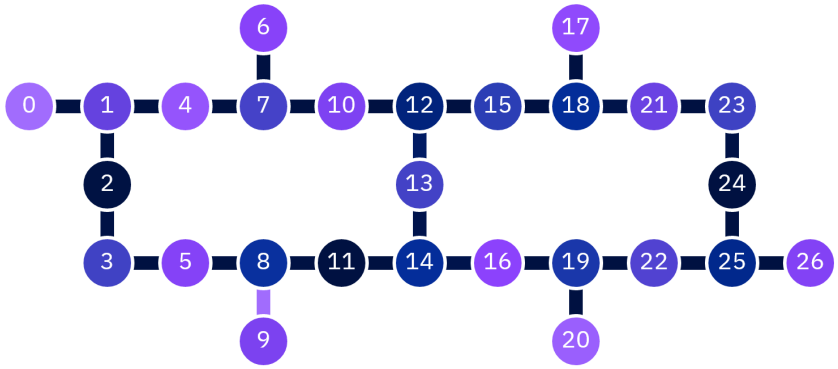

Figure 4: The qubit topology of the IBM-Q Mumbai system (source: [3]).

execute a two-qubit gate on an arbitrary pair of qubits. However, the specific problem that is to be solved may require operations on a large number of different qubit pairs which exceeds the number of directly connected qubit pairs available on current quantum systems.

In order to ensure that such problems can still be mapped onto current quantum systems, the application of two-qubit gates on an arbitrary qubit pair needs to be made possible. This can be achieved by creating indirect connections between qubits via the use of swap gates [26]. The structure of a swap gate was explained in Section 3.2. However, creating these indirect connections is costly, since the insertion of additional gates increases the depth of the quantum circuit and therefore increases the likelihood of gate errors and decoherence errors.

For IBM-Q systems, the process of mapping the original circuit for the problem onto the qubit topology of the quantum system is one step of the circuit compilation flow. An overview on this compilation flow is provided in [27]. Other steps include a transformation of the original gates in the circuit to the limited, yet universal gate set utilized by the IBM-Q systems as well as further optimizations that reduce the depth of the circuit.

Finally, detailed information about the remotely accessible IBM-Q systems can be found in [3]. More specifically, calibration data providing information on the average gate errors, readout errors, coherence times, average gate execution times among others is shown for each available system. The current systems offer up to 65 qubits and feature different qubit topologies. In addition to real quantum systems, various classical simulators can be accessed. For instance, the qasm simulator is a general simulator which is able to simulate up to 32 qubits. Other simulators feature more qubits, but can only be used for quantum circuits with specific properties.

3.6.2 Quantum Annealing

This section will provide an overview on the properties of current quantum annealers. Quantum annealing can be considered the application of adiabatic quantum computing on noisy systems [4]. Due to this noise, further restrictions need to be considered for quantum annealing. As mentioned in Section 3.5, adiabatic quantum computing has been shown to be universal

much like gate-based quantum computing. However, this argument only holds true when assuming noiseless qubits and it is still an open question whether or not quantum annealing can truly offer computational speedups that are not possible with classical computing [4].

The currently available quantum annealers offer significantly more qubits when compared to the current gate-based NISQ devices. For instance, the D-Wave Advantage system, which is the latest D-Wave system at the time of writing, has over 5,000 physical qubits [8]. However, in order to solve an optimization problem formulated as an Ising Hamiltonian on such a system, it first needs to be mapped onto the topology of the quantum annealer [28].

This is not a trivial task, since D-Wave systems, much like IBM-Q systems, have only limited connectivity graphs. Details about the architecture of quantum annealers, specifically about the so-called Chimera topology, are explained in [28]. In order to achieve a simple design and operation of the system, the D-Wave quantum annealing systems are made up by unit cells which are arranged in arbitrary structures.

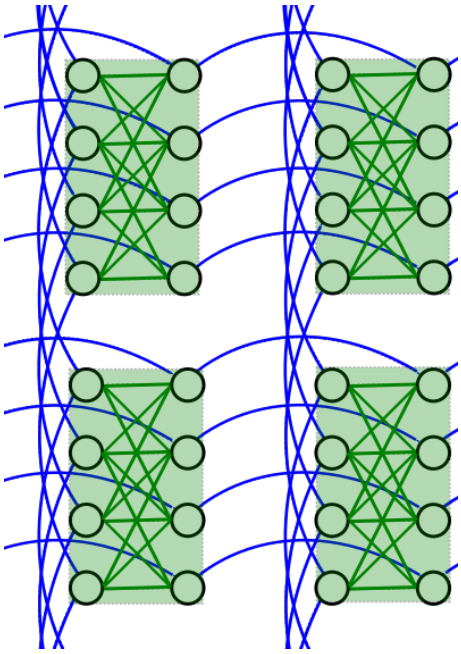

Figure 5: The arrangement of 32 qubits into 4 Chimera unit cells (source: [29]).

Figure 5 illustrates how 32 qubits are arranged into four Chimera unit cells, each of them containing eight qubits. The qubits are connected via couplers. It can be seen that each qubit inside of a unit cell is connected to each of the four qubits on the opposite side whereas no connection exists for a pair of qubits on the same side. Moreover, each qubit is connected to two other qubits which are included in vertically or horizontally adjacent unit cells. As such, each qubit is connected to at most six other qubits in a Chimera topology.

Chimera is the underlying qubit topology of the D-Wave 2X system which was used for solving MQO problems in [9]. However, in recent years, new systems such as the above-mentioned

Advantage system have been made available that feature an improved topology referred to as the Pegasus topology. In the Pegasus topology, 15 couplers exist per qubit in contrast to the Chimera topology, where one qubit is connected to only six other qubits as shown above [8].

However, some problems may require more than 15 connections for a single qubit to other qubits or simply connections between qubits in a way that is not supported by the topology for single qubits. In order to enable the embedding of problems with a more complex structure, a logical qubit is typically represented by a chain consisting of multiple physical qubits [28]. As such, when compared to the possible couplings between single qubits, a larger number of connections between these chains of qubits can be achieved. However, this means that oftentimes, the number of logical qubits that remain after the embedding process is significantly lower than the number of physical qubits offered by the system.

Since the problem of finding such an embedding is NP-complete, heuristic algorithms are typically used for this task [30]. One such algorithm is the *minorminer* algorithm, which is well-known and moreover used in the D-Wave Ocean software development kit (SDK) [30], [31]. Finding potentially better heuristic algorithms for this embedding process has been a topic of recent research. For instance, integer programming techniques are proposed in [32] as an alternative approach.

4 Background on Query Optimization

This chapter will give an overview on the two classical query optimization problems considered for this work. First, Section 4.1 will give an explanation on MQO. Next, Section 4.2 will provide an overview on the join ordering problem.

4.1 Multi Query Optimization

This section will first describe the general goal of MQO and will then explain the definition of an MQO problem used in this work. Typically, the result of a database query can be generated by using one out of several alternative plans. Each of these plans has its own associated execution cost. When only the resulting cost of generating one query result is taken into account, choosing the plan with the lowest associated cost is, for obvious reasons, the optimal strategy to generate the result.

However, as explained in [12], [13], when considering the global scenario of executing a batch of multiple queries, it might be possible to achieve a lower total cost for the query batch by using a different strategy. This strategy considers cost savings that arise when a subexpression that was generated for one query can be reused for another query in the batch, instead of evaluating the same subexpression multiple times. Choosing locally non-optimal plans can therefore potentially result in a lower total cost if these common subexpressions allow significant cost savings via reusage. Determining the set of plans that leads to the globally lowest cost by taking such cost savings into account is the goal of MQO.

Problem Definition. This work uses the definition of an MQO problem as given in [9], which retains the NP-hard property. Following that definition, the input of an MQO problem consists of a set of queries Q and a set of alternative plans P. Moreover, the set P_q with $P = \bigcup_q P_q$ includes all alternative plans corresponding to query $q \in Q$. Furthermore, the input includes a cost value c_p for each plan p denoting the cost of generating a query result when using the respective plan. Finally, the cost savings enabled by sharing subexpressions between the plans $p1$ and $p2$ are denoted by $s_{\mathrm{p1,\ p2}} > 0$.

Further following [9], a solution to the MQO problem is a set $P_e \subseteq P$ that contains all plans that are selected to be executed. A solution is only valid if for each query q, exactly one of its associated plans $p \in P_q$ is selected. A solution is optimal if the accumulated execution cost c_e, given by

$$c_e = \sum_{p \in P_e} c_p - \sum_{\{p1,p2\} \subseteq P_e} s_{\mathrm{p1,\ p2}}, \tag{25}$$

is minimized.

Table 1: An example MQO problem with three queries and eight plans in total. The associated execution costs merely illustrate the relative cost differences between the plans in this exemplary scenario and do not represent any real cost metric.

Query ID	Plan ID	Execution cost
1	1	10
1	2	12
1	3	15
2	4	9
2	5	16
3	6	7
3	7	12
3	8	9

Example. Table 1 shows an example MQO problem with three queries and eight execution plans in total. Moreover, it includes the respective execution cost for each plan.

Table 2: The possible cost savings for the MQO example and their associated plans which are to be used in combination.

Plan 1	Plan 2	Cost savings
2	4	4
2	8	5
3	4	6
5	7	7
5	8	3

In addition, Table 2 shows possible cost savings when selecting both of the corresponding plans. In order to illustrate the MQO problem, the resulting accumulated cost value for choosing locally optimal plans can be compared to the lowest accumulated cost value when possible cost savings are accounted for. Regarding the former, the resulting total cost is $10 + 9 + 7 = 26$ by selecting the plans 1, 4 and 6. In contrast, if the cost savings shown in Table 2 are also considered, the lowest accumulated cost is $12 + 9 + 9 - 4 - 5 = 21$ when choosing the plans 2, 4 and 8.

4.2 Join Order Optimization

This section will give a brief overview on the join ordering problem. As implied by the name, the goal of this query optimization approach is to determine the optimal order in which multiple database relations should be joined, which is a highly important optimization problem since determining a suitable join order has a high impact on the evaluation cost of a query [10]. More specifically, a join ordering problem is classified by a query graph, which serves as the input

to the problem, the types of join trees that are considered as well as the cost function that is applied [10]. The definitions for these three components will be described in the remainder of this section, following the explanations given in [10].

Query Graph. Based on these explanations, a query graph can be formally defined as a graph $G = (V, E)$ where V is the set of nodes and E is the set of edges. A node $v_i \in V$ with $i \in \mathbb{N}, 1 \leq i \leq |V|$ represents the relation R_i to be joined whereas an edge $e_{ij} \in E$ is labeled by a join predicate p_{ij}. Moreover, each join predicate corresponds to a selectivity f_{ij}, which is defined as

$$f_{ij} = \frac{|R_i \bowtie_{p_{ij}} R_j|}{|R_i| \cdot |R_j|}, \tag{26}$$

where $|R_i|$ denotes the cardinality of relation R_i. As such, the selectivity denotes the percentage of surviving tuples in relation to the cardinality of the cartesian product of R_i and R_j after applying the corresponding join predicate.

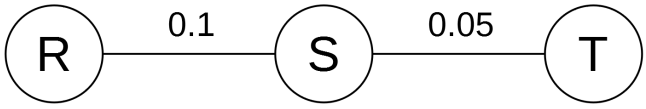

Figure 6: An example query graph for 3 relations.

Figure 6 depicts a small example query graph for 3 relations R, S and T. Note that the predicate label has been omitted so that the edges merely show the selectivities when applying the respective predicates. Depending on the query optimization approach, cross products may be considered as well. In this case, the cross product can be interpreted as a join operation with selectivity 1.

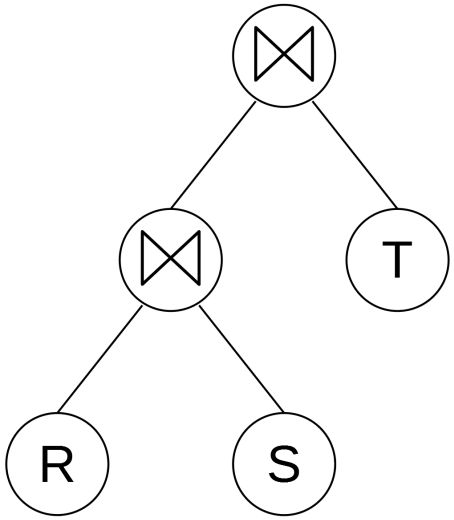

Figure 7: A possible join tree for the exemplary join ordering problem.

Join Tree. A solution to the join ordering problem can be represented by a join tree (also referred to as processing tree). Figure 7 depicts one possible join tree for the query graph presented in Figure 6. The relations are represented by the leaf nodes while the intermediate nodes denote the individual join operators. Typically, join ordering algorithms only take certain types of join trees into account in order to limit the search space, since the number of possible trees for some tree types, like left-deep trees, can be far smaller than the number of possibilities for other types such as unrestricted bushy trees.

The tree depicted in Figure 7 shows a left-deep join tree, a type which historically has been of high interest regarding query optimization research. The restriction to left-deep join trees means that for any join operator, only one join operand (represented by an outer node) can be an intermediate result, the remaining operand (represented by an inner node) needs to be one of the input relations.

The approach suggested in [16], in which the join ordering problem is solved as an MILP problem and which will be relevant for this work, also considers left-deep join trees. For this kind of join tree, the assignment of the relations for the leaf nodes determines the join order and can be expressed as a simple permutation. Following from that, the size of the search space for left-deep join trees is given by $n!$, where n denotes the number of relations.

Cost Function. Finally, in order to evaluate the quality of a solution, a join tree needs to be assigned a cost value. This cost value is produced by a cost function that takes a join tree as an input. As a result, the cost of processing a join tree depends on the cost function. Different cost functions are discussed in [33]. Out of these cost functions, the one that is most relevant for this work, is given by

$$C_{\text{out}}(n_i, n_j) := n_i n_j f_{i,j}. \tag{27}$$

This cost function is applicable for a single join of two relations i and j. Furthermore, n_i and n_j denote the cardinalities of the relations i and j respectively and $0 \leq f_{i,j} \leq 1$ denotes the selectivity of the join predicate. Further following [33], applied to a solution of the join ordering problem s (more specifically, to a permutation of the relations representing the join tree) over n relations, the cost metric is given as follows:

$$C(s) := \sum_{i=2}^{n} C_{\text{out}}(|s_1 ... s_{i-1}|, |s_i|), \tag{28}$$

where $|s_1 ... s_{j-1}|$ describes the cardinality of the intermediate result after joining relations 1 to $i-1$ and $|s_i|$ denotes the cardinality of relation i. Compared to the other cost functions shown in [10], [33], this cost function is rather basic and aims to minimize the cardinalities of the intermediate join results. However, as shown in [33], even when using the more basic cost function shown above, the general join ordering problem is still NP-complete. More details on the decision to use this cost function are given in Section 6.1.2.

Example. Finally, the following gives a demonstration of the join ordering problem, taking the example query graph shown in Figure 6 as an input. Let the cardinalities for the relations R, S and T be 10, 1000 and 1000 respectively. The query graph contains two edges representing join predicates. The respective predicate selectivities correspond to the ones contained in the labels for the edges, meaning there is a join predicate relating to the relations R and S with a selectivity $f_{\mathrm{RS}} = 0.1$ and a second predicate for the relations S and T with a selectivity $f_{\mathrm{ST}} = 0.05$. As the example query graph only includes 3 relations, it is clear that all resulting join trees are classified as left-deep trees. Finally, let the cost function be the one given in Equation 28.

Table 3: The cost calculation for each possible join order for the example query graph.

Join order	**Cost calculation**	**Resulting cost**
$(R \bowtie S) \bowtie T$	$C_{\mathrm{out}}(\lvert R\rvert, \lvert S\rvert) + C_{\mathrm{out}}(\lvert RS\rvert, \lvert T\rvert) = \lvert R\rvert\lvert S\rvert f_{\mathrm{RS}} + \lvert RS\rvert\lvert T\rvert f_{\mathrm{ST}} =$ $= 10 \cdot 1{,}000 \cdot 0.1 + 1{,}000 \cdot 1{,}000 \cdot 0.05 = 1{,}000 + 50{,}000$	51,000
$(R \bowtie T) \bowtie S$	$C_{\mathrm{out}}(\lvert R\rvert, \lvert T\rvert) + C_{\mathrm{out}}(\lvert RT\rvert, \lvert S\rvert) = \lvert R\rvert\lvert T\rvert + \lvert RT\rvert\lvert S\rvert f_{\mathrm{RS}} f_{\mathrm{ST}} =$ $= 10 \cdot 1{,}000 + 10{,}000 \cdot 1{,}000 \cdot 0.1 \cdot 0.05 = 10{,}000 + 50{,}000$	60,000
$(S \bowtie T) \bowtie R$	$C_{\mathrm{out}}(\lvert S\rvert, \lvert T\rvert) + C_{\mathrm{out}}(\lvert ST\rvert, \lvert R\rvert) = \lvert S\rvert\lvert T\rvert f_{\mathrm{ST}} + \lvert ST\rvert\lvert R\rvert f_{\mathrm{RS}} =$ $= 1{,}000 \cdot 1{,}000 \cdot 0.05 + 50{,}000 \cdot 10 \cdot 0.1 = 50{,}000 + 50{,}000$	100,000

Table 3 shows the resulting costs for each join tree that can be produced for the example query graph depicted in Figure 6. Note that the order in which the first two relations are joined does not make a difference for the used cost function. As such, Table 3 does not include these join orders in order to avoid superfluous entries. Also, while the used cost function as stated in [33] does include the costs for the final join, those are not actually needed for determining the optimal join order as they are obviously the same for all possible join orders. However, for the purpose of illustrating the problem, the final join has been included in the calculations as well.

It becomes clear that choosing a suitable join order makes a big difference even for the small example query graph which only consists of three relations. The difference is particularly striking when comparing the cost for the optimal join order $(R \bowtie S) \bowtie T$ with the one for the worst case order $(S \bowtie T) \bowtie R$. This underlines the importance of generating suitable join orders as a part of query optimization.

5 Solving Multi Query Optimization with Quantum Computing

This chapter will explain the quantum computing approach for MQO problems. In Section 5.1, an existing approach for formulating an MQO problem as a QUBO problem will be described. Next, in Section 5.2, frameworks used for the implementation as well as details about the implementation will be explained. Finally, an evaluation of the applicability of current quantum systems with regards to MQO will be given in Section 5.3. More specifically, the existing results for a D-Wave quantum annealing system will be compared with simulation results for a current IBM-Q machine.

5.1 QUBO Formulation

An existing approach to solve MQO problems with quantum computing has been presented in [9]. More specifically, the authors solved a variety of MQO problems using quantum annealing on the D-Wave 2X system. As explained, quantum annealing can be applied on Ising Hamiltonians, which can moreover be expressed as QUBO problems. A method for formulating an MQO problem as a QUBO problem has been shown in [9].

However, QUBO problems can not only be used as an input for quantum annealing. Optimization algorithms for gate-based quantum systems, such as QAOA and VQE, which have been discussed in Section 3.4, also take Ising Hamiltonians or QUBO problems as an input. Therefore, the method shown in [9] can also be used to solve MQO problems on IBM-Q systems. A comparison of the results of the quantum annealing approach for D-Wave systems as presented in [9] and current IBM-Q systems in regards to the possible MQO problem dimensions will be given in Section 5.3.

Energy Formula. The definition of an MQO problem used for this work was described in detail in Section 4.1. The remainder of this section will explain the approach to formulate MQO problems as QUBO problems as proposed in [9]. First, the overall QUBO formulation, also referred to as *energy formula* in [9], is given by

$$E = \omega_L E_L + \omega_M E_M + E_C + E_S. \tag{29}$$

It can be seen that the QUBO formulation is divided into four terms. A term either ensures that minimizing the energy cost produces valid results or it contributes in encoding the optimization goal. The former includes the terms E_L and E_M, which are additionally multiplied by the weights ω_L and ω_M respectively. Combined with these weights, sometimes also called energy penalties, these two terms ensure that solutions which are not valid lead to very high energy levels. Therefore, a solution that minimizes the energy cost is always valid if the weights are set correctly.

Further following [9], E_L is given by

$$E_L = -\sum_{p \in P} X_p, \tag{30}$$

where P is the set of overall plans and $X_p \in \{0, 1\}$ equals 1 if plan p is selected to be executed and 0 otherwise. As the term reduces the overall energy due to a negative factor, it incentivizes that a plan p is selected rather than not selected. This ensures that at minimum, one plan per query is selected. However, in order for an MQO solution to be valid, exactly one plan per query needs to be executed. This is achieved by additionally including the term E_M, which is given by

$$E_M = \sum_{q \in Q} \sum_{\{p1,p2\} \subseteq P_q} X_{p1} X_{p2}, \tag{31}$$

where Q is the set of queries and P_q is the set of alternative plans for query q. This encoding ensures that E_M is greater than 0 once more than a single plan is selected for query q, in which case the penalty weight ω_M will be added to the overall energy. Assuming ω_L and ω_M are both set to suitable values (as explained below), minimizing the energy will lead to valid solutions.

The remaining terms are import for finding optimal solutions. However, their impact on the overall energy is comparatively lower as they lack weight factors. As a result, valid solutions with high costs are preferred over invalid solutions with low costs. The term E_C ensures that a higher execution cost leads to a higher energy and is given by

$$E_C = \sum_{p \in P} c_p X_p, \tag{32}$$

where c_p denotes the costs of executing plan p. Finally, in order to take possible savings into account, E_S is given by

$$E_S = -\sum_{\{p1,p2\} \subseteq P} s_{\mathrm{p1,\ p2}} X_{p1} X_{p2}, \tag{33}$$

where $s_{\mathrm{p1,\ p2}}$ are the cost savings that are possible when executing both plans $p1$ and $p2$. This term ensures that savings will only be subtracted from the overall energy if both of its related plans $p1$ and $p2$ are selected.

Penalty Weights. Finally, as mentioned above, choosing suitable values for the weights ω_L and ω_M is essential in order for the terms E_L and E_M to have their intended effect. The weight of a term has to be set in relation to the possible energy impact of the other terms in the Hamiltonian. This is done for ω_L by setting it to a value so that the inequality

$$\omega_L > \max_{p \in P} c_p \tag{34}$$

holds. This enforces that selecting a plan always leads to a lower overall energy than not selecting it to avoid its cost contribution to the overall energy. However, in order to not incentivize the execution of all plans for all queries, the remaining weight ω_M must be set accordingly as well. More specifically, the following inequality must hold:

$$\omega_M > \omega_L + \max_{p1 \in P} \sum_{p2 \in P} s_{\mathrm{p1,\,p2}}. \tag{35}$$

First, by setting $\omega_M > \omega_L$, this inequality ensures that the energy cost of selecting more than one plan per query is higher than the energy savings achieved through E_L by selecting more than one plan. However, without any additional constraints, it would still be possible that selecting more than one plan per query leads to a lower energy due to possible additional savings coming from the term E_S. This is prevented by adding the largest possible savings value achievable by one plan to the right side of the inequality.

5.2 Implementation

This section will provide an overview on the implementation of the quantum computing approach for MQO. First, Section 5.2.1 will describe the frameworks used for the implementation whereas Section 5.2.2 will explain implementation details.

5.2.1 Frameworks Used

For the implementation of the quantum computing approach for MQO for IBM-Q systems, the Qiskit SDK has been used [34]. Qiskit enables the creation and execution of quantum circuits either locally via simulators or remotely on IBM-Q machines, where both simulators or real quantum processors can be run. Currently, only real quantum systems with up to five qubits can be freely accessed. As such, the implementation makes use of the simulators in order to solve MQO problem instances of more relevant dimensions. In this work, Qiskit version 0.27.0 has been used. Moreover, as recommended in the Qiskit documentation, the Anaconda Python distribution (version 4.9.2) has been used to run Qiskit code via Jupyter [35].

Furthermore, the Qiskit SDK also offers implementations for the hybrid quantum-classical algorithms, specifically VQE and QAOA (which were discussed in Section 3.4), that have been utilized. Finally, the docplex library (version 2.20.204) has been used for modeling and formulating the MQO problem [36], which can then be easily converted to the problem representation required by the hybrid quantum-classical algorithms using the qiskit_optimization package (version 0.1.0).

5.2.2 Implementation Details

This section will explain the steps needed to solve an MQO problem on either an IBM-Q simulator or a real quantum system. Typically, for optimization problems of this kind, most

of the development time is spent determining suitable reformulations. In contrast, the actual implementation size in terms of lines of code is limited, which is the case for solving MQO with quantum computing [37]. In the first step, the DOCPLEX `Model` class is used for the problem formulation. First, for each plan contained in the problem input, one binary variable is added to the `Model`. Afterwards, the four QUBO terms discussed in Section 5.1 are formulated as mathematical expressions. They are then added to the `Model` and their sum is specified as the minimization objective of the `Model`.

Next, the DOCPLEX `Model` can be converted into an object of the `QuadraticProgram` class using the `from_docplex()` method. This object serves as an input for the `solve()` method provided by the `MinimumEigenOptimizer` class. This class is particularly useful for using the hybrid quantum-classical algorithms, as it converts the `QuadraticProgram` object to a QUBO problem. Moreover, when initializing the `MinimumEigenOptimizer`, an instance of either algorithm class can be provided as a solver, which is used by the `MinimumEigenOptimizer` to find an optimal solution for the QUBO problem.

When initializing the VQE or QAOA algorithm, a backend needs to be provided in order to specify in which way the circuit execution should proceed. It is possible to pass a backend for local simulations or to specify a remote IBM-Q system or a remote simulator as a backend. Details on the simulation backend used for this work will be provided in Section 5.3.2. For QAOA, the initial point has been initialized using zeros. Apart from this, both VQE and QAOA have been initialized using the default parameters as specified by the QISKIT implementation for the algorithms. As such, the number of operator repetitions p for QAOA is 1. Higher values for p quickly lead to large circuit depths even for small problems. After the execution of the algorithm has concluded, the optimization result is returned by the `MinimumEigenOptimizer`. Moreover, the optimal QAOA or VQE quantum circuit is retrieved via the `get_optimal_circuit()` method of the respective quantum algorithm object.

The depth of the resulting quantum circuit plays an important role when evaluating the applicability of the quantum computing approach, as will be discussed in Section 5.3. Instead of real quantum systems, simulators, which per default operate based on an optimal qubit topology (i.e., each qubit is connected to every other qubit), are used in this work. As such, the depth of the returned quantum circuits does, without further steps, not correspond to the depth on a real system.

In order to determine the depth of a quantum circuit that can be executed on such a real system, the `transpile()` function, which is included in the QISKIT.COMPILER package, is used to transform the quantum circuit into one that conforms to a specified topology representing a real system. For the `transpile()` function, the default parameter settings have been used. As such, out of the four circuit optimization levels 0 to 4, level 1 is used, which means light optimization is performed on the circuits. More details on the utilized qubit topology will be given in the next section.

5.3 Evaluation

This section will evaluate the applicability of the quantum-classical algorithms on IBM-Q systems in comparison to quantum annealing on a D-Wave system described in [9]. In Section 5.3.1, the scaling behavior of the required number of qubits will be described. Moreover, the impact of certain properties of the MQO problem on the number of quadratic terms in the QUBO matrix will be explained. Next, in Section 5.3.2 the applicability of the two hybrid quantum-classical algorithms on MQO using IBM-Q simulators will be evaluated.

5.3.1 Scaling Behavior of the Qubits and the Quadratic Terms

Each plan in the MQO problem is represented by one logical variable in the QUBO problem [9]. As such, the number of plans is a lower bound for the number of required qubits. However, for D-Wave systems, a chain of physical qubits is used to represent one logical variable in order to enable the embedding of more complex problems. The D-Wave 2X system used in [9] has over 1,000 physical qubits. However, due to the use of qubit chains, the number of variables the input problem may contain is typically significantly lower, depending on the specific problem. This is not the case for IBM-Q systems, for which one physical qubit is sufficient to represent one variable.

In [9], the D-Wave 2X system was able to solve different classes of MQO problems with varying numbers of plans per query (PPQ). The class with the lowest number of PPQ included 1,074 plans while the class with the highest number of PPQ contained 540 plans. In contrast, the largest currently available IBM-Q system in terms of qubit numbers offers 65 qubits. Therefore, the upper bound for the number of plans is significantly lower on IBM-Q systems available at the time of writing compared to the D-Wave 2X system.

Taking into account the newest D-Wave machine at the time of writing, which is the D-Wave Advantage system offering over 5,000 qubits and having an improved qubit connectivity [8], the MQO problem dimensions that can be solved on D-Wave systems have likely increased significantly. This further widens the gap between current IBM-Q and D-Wave systems regarding their applicability in regards to MQO problems. In summary, D-Wave systems can be used to solve significantly larger MQO problems when compared to current IBM-Q systems.

The number of qubits is not the only limiting factor that needs to be considered for IBM-Q systems. As explained in a previous section, the number of quadratic terms in the QUBO matrix also influences the applicability of both quantum annealing and the gate-based QAOA algorithm.

In case of the former, the task of mapping the logical QUBO matrix onto the structure of the D-Wave machine becomes more complex due to the higher number of required connections. For the latter, the depth of the quantum circuit increases due to the higher number of indirect qubit connections. The simulation results illustrating the concrete negative impact of an increase in

the number of quadratic terms on the applicability of two simulated quantum algorithms for IBM-Q systems will be presented in Section 5.3.2.

With regards to the QUBO formulation of an MQO problem, quadratic terms appear in the energy formulas E_M and E_S whereas E_L and E_C only contain linear terms. More specifically and as explained in [9], E_M requires connections between each pair of alternative plans for a query while E_S does so for plans that share results. The impact of the former has been highlighted by the results for different MQO problem classes with varying numbers of PPQ presented in [9]: The total number of plans the MQO problem may contain in order for it to still be solvable on quantum annealers decreases with an increasing number of PPQ.

5.3.2 Evaluation for IBM-Q systems

In the previous section, we explained how a lower bound for the required qubits corresponds to the number of plans in the MQO problem. For D-Wave systems, depending on the density of the QUBO matrix, additional physical qubits are needed to represent all logical variables. Even so, in comparison to current IBM-Q systems, they are still capable of solving significantly larger problems despite this overhead in required physical qubits. However, in the previous section, only the number of qubits was taken into consideration when analyzing the applicability of IBM-Q systems.

This section will explain further limitations for IBM-Q devices by showing how the density of the QUBO matrix also impacts the applicability of these systems. More specifically, the impact of an increasing number of quadratic terms on the quantum circuit depths resulting from the QAOA algorithm will be analyzed. In addition, the QAOA circuit depths will be compared to the circuit depths for the VQE algorithm, which do not depend on the number of quadratic terms. The quantum algorithms are applied by using the remote IBM-Q qasm simulator. Moreover, results for simulations that are based on a state-of-the-art topology will be compared to ones for the optimal qubit topology provided by the qasm simulator. Detailed information about the specifications of the simulators and real systems can be found in [3].

The qasm simulator assumes an optimal qubit topology where each simulated qubit is connected to all other qubits. In contrast, direct qubit connections are very sparse on state-of-the-art topologies. The IBM-Q Mumbai system, whose qubit topology is shown in Figure 4, was used as the state-of-the-art system for the simulations. As explained in an earlier section, connections between qubits which are not directly connected need to be established indirectly via the use of swap gates as a part of the transpilation process. However, this leads to an expansion of the depth of the resulting quantum circuit.

QAOA Circuit Depths. Figure 8 shows two charts presenting simulation results for the QAOA algorithm. For both charts, the x-axis denotes the total number of plans in the MQO problem whereas the y-axis denotes the depth of the optimal quantum circuit returned by the

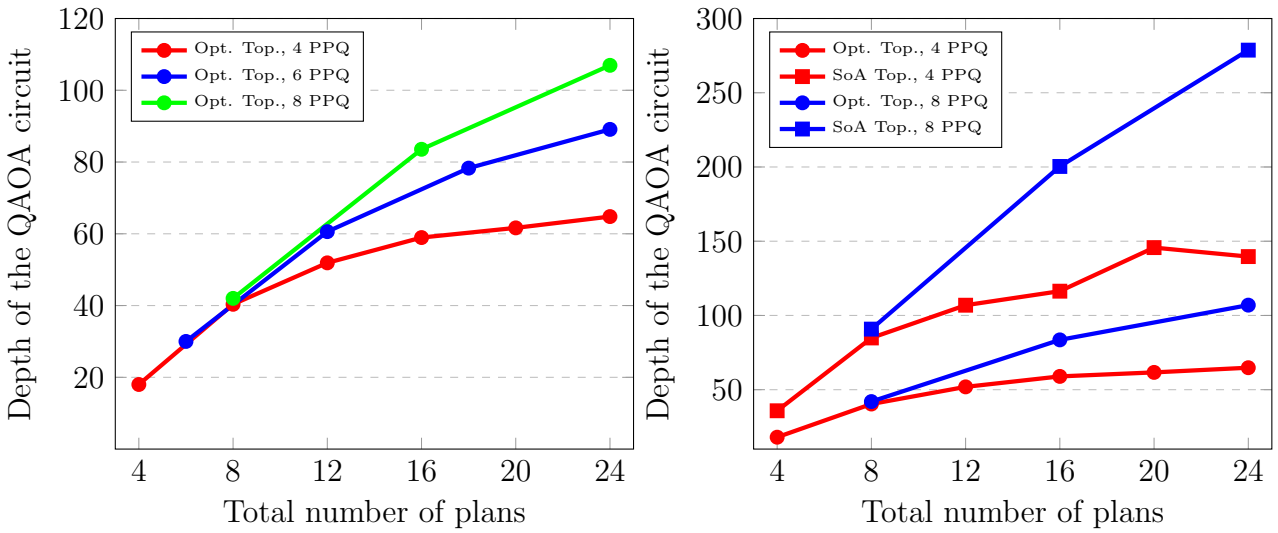

Figure 8: The MQO circuit depths for QAOA with respect to varying qubit numbers, plans per query, and qubit topologies.

algorithm. Simulations have been conducted for randomly generated MQO instances considering up to 24 plans. Moreover, the charts depict mean circuit depths over 20 randomly generated MQO instances.

The left chart depicts three graphs for varying numbers of PPQ. Furthermore, it shows simulation results when using an optimal qubit topology. It can be seen that, as the number of PPQ increases, so does the resulting circuit depth. Comparing the mean circuit depth when the MQO problem has 24 plans in total for 4 PPQ with the one for 8 PPQ, the latter is approximately 65% larger.

The right chart likewise depicts simulation results for varying numbers of PPQ, however, it also shows graphs for different qubit topologies including ones for the Mumbai state-of-the-art topology. Comparing the circuit depth increases resulting from the usage of the Mumbai topology for 4 PPQ and 8 PPQ at 24 plans in total, it is clear that the impact of using a non-optimal topology becomes more drastic for denser QUBO matrices. For 4 PPQ, using the Mumbai topology increases the mean circuit depth by approximately 116% at 24 plans in total, whereas it increases by roughly 160% for 8 PPQ.

QAOA Versus VQE. In addition to QAOA, simulations have also been conducted for the VQE hybrid quantum-classical algorithm. Figure 9 depicts two charts that show simulation results for both algorithms. More specifically, the left chart depicts the results for the optimal qubit topology of the qasm simulator whereas the right chart shows results for the Mumbai topology. Moreover, both charts only show different graphs for varying numbers of PPQ with respect to QAOA. For VQE, only the total number of plans in the MQO problem influences the depth of the resulting quantum circuit whereas the density of the QUBO matrix does not

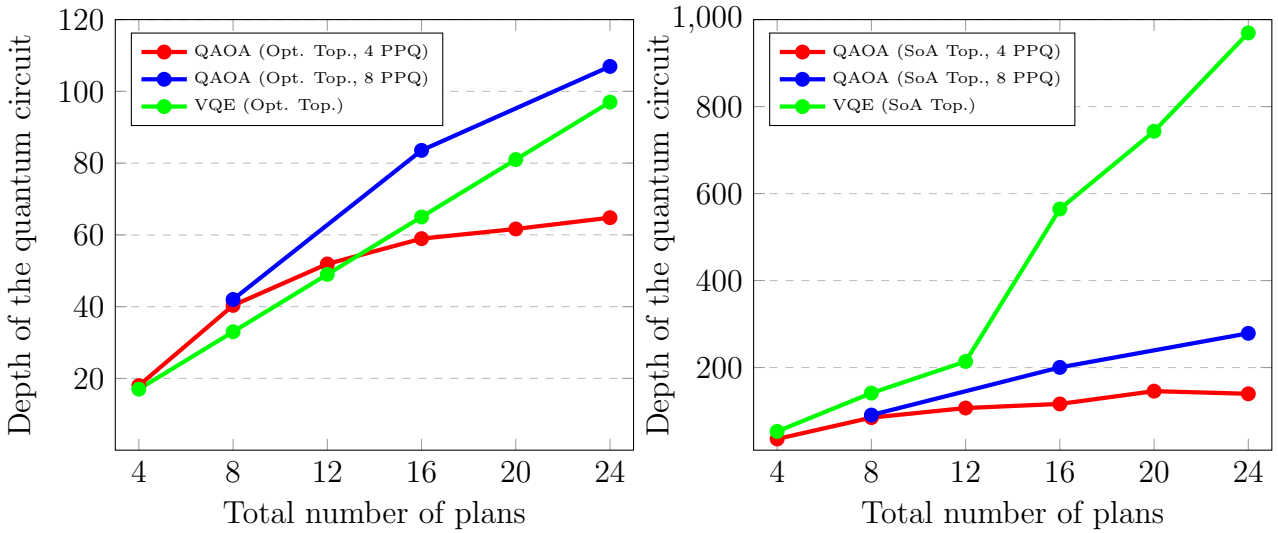

Figure 9: MQO circuit depths for VQE and QAOA and varying qubit topologies.

play a role.

Regarding the results for the optimal qubit topology, it can be seen that for both QAOA graphs, the rate at which the mean circuit depth grows decreases with an increasing number of plans. In contrast, the graph for VQE grows linearly with the number of plans. As such, as the total number of plans further increases beyond the numbers shown in the charts, the use of VQE will likely lead to larger circuit depths when compared to using QAOA on MQO problems with 4 or 8 PPQ.

However, when considering the results for the Mumbai topology, which are depicted in the right chart, it becomes clear that mapping the VQE circuit onto a state-of-the-art topology leads to a drastic increase of the circuit depth. For the optimal topology, the resulting VQE circuit depth for 24 plans in total is 97, whereas the mean circuit depth over 20 circuit transpilations is around 970 when using the state-of-the-art topology, thus increasing the depth by roughly 900%. This overhead in circuit depth is significantly larger when compared to QAOA, where the usage of the state-of-the-art topology leads to an increase of around 160% for 8 PPQ and 116% for 4 PPQ when the MQO problem contains 24 plans.

Circuit Depth and Coherence Times. In order to evaluate the feasibility of executing the quantum algorithms with regards to their corresponding mean circuit depths on real systems in a more concrete way, the specifications of the system should be considered. More specifically, the average coherence time as well as the average time it takes for a gate to be executed are relevant properties. As explained in [26], the coherence time affects the probability of errors due to decoherence depending on the time it takes to execute the circuit. Moreover, two separate coherence times, T1 and T2, need to be considered.

As described in [26], T1 refers to the time it takes for a qubit in the high-energy state $|1\rangle$ to

naturally decay to the low-energy state $|0\rangle$. On the other hand, T2 is connected to the time it takes until the superposition state $(|0\rangle + |1\rangle / \sqrt{2})$ decays to $|0\rangle$ or $|1\rangle$. More specifically, as described in [25], both T1 and T2 are constants calibrated for the quantum system so that the probability of an error p_{err} in relation to the respective coherence time T is proportional to

$$p_{\text{err}} = 1 - e^{-t/T}, \tag{36}$$

where t is the computation time. As such, once the execution time of the quantum circuit reaches the coherence time, the probability for an error due to decoherence is approximately $p_{err} = 1 - e^{-1} \approx 0.63\%$, increasing further as the computation goes on. Based on the two coherence times of the Mumbai system, which, at the time of writing, are 117.22 µs and 118.47 µs for T1 and T2 respectively, and the average gate time g_{avg} given by 471.111 ns, the maximum circuit depth d_{max} that can be executed within the coherence time can be calculated:

$$d_{max} = \left\lfloor \frac{\min(\text{T1}, \text{T2})}{g_{avg}} \right\rfloor = \left\lfloor \frac{117{,}220 \text{ ns}}{471.111 \text{ ns}} \right\rfloor = 248. \tag{37}$$

Taking this circuit depth as a threshold after which a reliable circuit execution becomes increasingly improbable due to decoherence errors, it can be seen that the VQE circuit depths beyond MQO problems with 12 plans in total are too large for reliable calculations on real systems, as the circuit depths for the Mumbai topology exceed the threshold circuit depth by a large margin. The same is true for the QAOA circuit for 8 PPQ and 24 plans in total, which also exceeds this threshold. In contrast, all circuit depths for 4 PPQ can be executed well within the coherence time.

Summary. In summary, it becomes clear that the number of quadratic terms in the specific QUBO problem imposes further limitations on the applicability of quantum computing for MQO on current systems. The set of MQO problems which can be reliably solved on current gate-based systems is already very restricted due to the limited number of available qubits. However, when also considering errors such as decoherence errors, the set of problems becomes further limited as highlighted by the simulation results for MQO problems with a higher number of PPQ.

6 Solving the Join Ordering Problem with Quantum Computing

This chapter shows how to solve the join ordering problem, which was described in Section 4.2, via the use of quantum computing. First, a two-step approach for transforming the join ordering problem into QUBO form is presented in Section 6.1. Next, implementation details for both IBM-Q and D-Wave machines are explained in Section 6.2 and finally, the results will be evaluated in Section 6.3.

6.1 QUBO Formulation

Figure 10: The steps of the join ordering reformulation approach.

This section will give a detailed explanation on a multi-step approach for solving join ordering problems on quantum processing units (QPU). Figure 10 shows the general steps involved. First, the initial join ordering problem can be formulated as an MILP problem. A brief overview on MILP problems is given in Section 6.1.1 and the actual reformulation approach is explained in Section 6.1.2. After some additional adjustments, which are described in Section 6.1.3, the join ordering MILP problem can be expressed as a binary integer linear programming (BILP) problem, which can be formulated as a QUBO problem as explained in Section 6.1.4. This allows us to ultimately solve join ordering problems on QPUs.

6.1.1 Mixed Integer Linear Programming

The first transformation step involves the formulation of the join ordering problem as an MILP problem. This section will give a short explanation on MILP problems before the actual transformation of the join ordering problem will be explained in the next sections. MILP problems are optimization problems with the goal of determining values for a set of variables such that the output of a specified objective function is either minimized or maximized. As explained in [16], reformulating problems as MILP problems and using state-of-the-art MILP solvers is considered promising since these solvers have matured over a long period of time.

As described in [38], an MILP problem is formally stated as follows:

$$\begin{aligned} \max \quad & cx + hy \\ \text{subject to} \quad & Ax + Gy \leq b \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

MILP problems may contain both integer and non-integer variables, which are given by $x \in \mathbb{Z}_0^n$ and $y \in \mathbb{R}_0^p$ respectively. Furthermore, $c \in \mathbb{R}^n$ and $h \in \mathbb{R}^p$ are the coefficient vectors for the objective function. Additionally, the problem contains m constraints which need to be satisfied by the variable assignment for a solution to be valid. Both A and G are coefficient matrices with A being a $m \times n$ and G being a $m \times p$ matrix. Finally, $b \in \mathbb{R}^m$ is a constraint vector.

6.1.2 Mixed Integer Linear Programming Formulation for the Join Ordering Problem

This section will describe how the join ordering problem can be formulated as an MILP problem based on the explanations given in [16]. More specifically, it is shown in [16] how to model the join ordering problem for finding left-deep join trees with the goal of minimizing the cost metric C_{out} shown in [33] and defined in Equation 28. Furthermore, [16] also considers cartesian products in addition to joins.

Preliminary Considerations. It is also described in [16] how to model more sophisticated cost metrics. However, those require additional variables, some of which are continuous and therefore require an even greater number of additional qubits. As the number of available qubits on current quantum computing machines is very limited, this work only utilizes the more basic model that minimizes the cost function given in Equation 28, which, as shown in [33], can be achieved by minimizing the cardinalities of the intermediate join results.

One problem that needs to be considered for the transformation into MILP form is given by the fact that the cost function as described in Equation 28 contains products, which cannot be modelled by MILP problems. As such, it is suggested in [16] to use logarithmic values for the intermediate cardinalities, making use of the fact that the logarithm of a product can be expressed as the sum of the logarithmic values of its factors.

In order to approximate the actual intermediate cardinalities based on the logarithmic ones, an arbitrary number of threshold values is used, as will be explained below. This approach introduces a trade-off: The more threshold values are used, the more accurate the solution becomes. However, more accurate solutions demand more binary variables. Due to the fact that every binary variable in the MILP model needs to be represented by one qubit and taking into account the limited number of qubits current quantum computing machines can offer,

this trade-off is even more significant when using quantum computing for solving join ordering problems with this method.

MILP Encoding. Further following [16], the MILP problem contains the following binary variables:

- tii_{tj}: Indicates whether or not relation t is contained in the inner operand of the j-th join
- tio_{tj}: Indicates whether or not relation t is contained in the outer operand of the j-th join
- pao_{pj}: Indicates whether or not the p-th predicate can be evaluated on the outer operand of the j-th join
- cto_{rj}: Indicates whether or not the logarithmic value of the r-th threshold value θ_r has been reached by the logarithmic cardinality of the outer operand of the j-th join

Since this work only considers the basic cost function which aims to minimize intermediate cardinalities, the objective function of the MILP problem is simply given by

$$\min \sum_r \sum_j cto_{rj} \delta\theta_r. \tag{38}$$

The term $\delta\theta_r$ is defined in such a way that a threshold value is not added several times to the objective function once it has been reached. One exemplary way suggested in [16] to achieve this is to define the value for $\delta\theta_r$ as $\theta_r - \theta_{r-1}$ for $r \geq 1$ and $\delta\theta_0 = \theta_0$. This definition for $\delta\theta_r$ is applied in this work.

Further following [16], the constraints of the following types have to be added to the MILP problem in order to enforce valid solutions to the underlying join ordering problem:

1. $\sum_t tio_{t0} = 1$.
2. $\forall j : \sum_t tii_{tj} = 1$.
3. $\forall j \forall t : tio_{tj} + tii_{tj} \leq 1$.
4. $\forall j \geq 1 \forall t : tio_{tj} = tii_{t,j-1} + tio_{t,j-1}$.
5. $\forall p \forall j : pao_{pj} \leq tio_{T_1(p)j}$, where $T_1(p)$ denotes the first relation the predicate p refers to.
6. $\forall p \forall j : pao_{pj} \leq tio_{T_2(p)j}$, where $T_2(p)$ denotes the second relation the predicate p refers to.
7. $\forall j \forall r : \sum_t \log(Card(t)) tio_{tj} + \sum_p \log(Sel(p)) pao_{pj} - cto_{rj} \cdot \infty_{rj} \leq \log(\theta_r)$, where $Card(t)$ is the cardinality of the relation t, $Sel(p)$ is the selectivity of the predicate p, ∞_{rj} is a sufficiently large constant for satisfying the constraint when cto_{rj} is set to 1 and θ_r is the r-th threshold value.

The assignment of the tio_{t0} and tii_{tj} variables expresses the solution to the join ordering problem, as it determines the leaves of the join tree and thus indicates the order in which the relations are joined. Constraints of types 1 and 2 ensure that exactly one relation may be chosen for any leaf while constraints of type 3 invalidate solutions where the same relation is part of both the inner and outer operand of the j-th join. Constraints of type 4 ensure that, once a relation has been an operand of a join, it will be contained in the outer operands of all subsequent joins and will thus contribute to the cardinality of the intermediate results.

Constraints of types 5 and 6 enforce that a predicate can only be applied to a join if both of the relations it refers to are already contained within the outer operand of the join. Finally, constraints of type 7 ensure that the indicator variable for the r-th threshold value is set to 1 for the j-th join if the logarithmic cardinality of the intermediate result (consisting of all relations that are contained in its outer operand) exceeds the logarithmic threshold value.

Example. The following illustrates the MILP formulation process for a small exemplary join ordering problem. Let the problem input include three relations A, B and C, each containing 10 tuples. Furthermore, let the problem input contain one join predicate p with a selectivity of 0.1 for the relations A and B. Moreover, let the input include one threshold value $v = 10$. Then, for each of the three relations and for each of the two joins, variables tii_{tj} and tio_{tj} with $0 \leq j \leq 1$ and t being a placeholder for the respective relation are introduced. In addition, for the single predicate p, two variables pao_{p0} and pao_{p1} are introduced for the first and second join respectively. Finally, the threshold indicator variables cto_{v0} and cto_{v1} are likewise introduced for the single threshold value v and both joins.

For this small example scenario, an optimal solution which minimizes intermediate cardinalities is obviously given by the join orders $(A \bowtie B) \bowtie C$ or $(B \bowtie A) \bowtie C$. For the first join, the logarithmic intermediate cardinality for the outer operand, which is $\log_{10}(10) = 1$, never exceeds the logarithmic threshold value $\log_{10}(10) = 1$. As such, the respective type 7 constraint for cto_{v0} is always satisfied without the threshold variable needing to be activated. For the second join, the logarithmic intermediate cardinality for the outer operand is given by $\log_{10}(10) + \log_{10}(10) + \log_{10}(0.1) = 1 + 1 - 1 = 1$ if variable pao_{p1} is activated. Then, the threshold variable cto_{v1} is also not activated, which leads to an optimal solution.

In accordance to constraint types 5 and 6, the predicate only influences the intermediate cardinality of the second join if the relations A and B are both contained in its outer operand, which requires $tio_{A1} = 1$ and $tio_{B1} = 1$. Otherwise, $pao_{p1} = 0$ and the intermediate threshold value becomes $\log_{10}(10) + \log_{10}(10) = 1 + 1 = 2$, which activates the threshold variable cto_{v1} due to the type 7 constraint. As a result, the corresponding value $\delta\theta_0 = 10$ is added to the objective function, which indicates that the accumulated intermediate cardinality for the non-optimal solution exceeded 10. For any solution, the join order can be inferred from the assignment for the variables tii_{tj} and tio_{t0}. For this specific example, the optimal join order $(A \bowtie B) \bowtie C$ is given by the variable assignment $tio_{A0} = 1$, $tii_{B0} = 1$ and $tii_{C1} = 1$.

6.1.3 Elimination of Inequality Constraints

Instead of using an MILP solver to determine an optimal solution, the MILP problem is further transformed into a form suitable for quantum computing for this work. Since the MILP formulation does not contain any non-binary variables, it is also a BILP problem, which can be transformed into an Ising Hamiltonian as shown in [20] and can thus be solved on current quantum computing devices. However, in order to bring the BILP problem into this form, every inequality constraint first needs to be converted to an equality constraint. This can be achieved by adding slack variables to the inequality constraints [38]. The following shows the transformed constraint of type 7, introducing a slack variable sl_{rj}:

$$\sum_t \log(Card(t))tio_{tj} + \sum_p \log(Sel(p))pao_{pj} - cto_{rj} \cdot \infty_{rj} + sl_{rj} = \log(\theta_r). \tag{39}$$

For any inequality constraint in the model other than those of type 7, one binary slack variable is sufficient to turn it into an equality constraint. As such, one additional qubit is necessary for those constraints.

However, for the seventh type, the slack variable sl_{rj} needs to be continuous, which is not possible for BILP problems. In order to circumvent this, the continuous slack variable can be approximated with several discrete variables. As shown in [39], an integer variable with an upper bound C can be expressed via $n = \lfloor \log_2(C) \rfloor + 1$ binary variables. Utilizing this method similarly to how it was done in [40], a continuous slack variable csl with an upper bound C can be approximated in the following way:

$$csl = \omega \sum_{i=1}^{n} 2^{i-1} bsl_i, \tag{40}$$

where $n = \lfloor \log_2(\frac{C}{\omega}) \rfloor + 1$, $\omega = 0.1^p$ with $p \in \mathbb{N}_0^+$ and $bsl_i \in \{0, 1\}$ denotes a binary slack variable. The factor ω determines the precision of the approximation and introduces another trade-off: the lower the value assigned to ω, the more precisely the continuous slack variable can be approximated. In return, however, a larger number of binary variables and thus a higher number of qubits is necessary. Explanations on upper bounds for the number of required binary variables will be given in Section 6.3.1.

6.1.4 Bringing the BILP Problem into QUBO Form

The previous sections gave an explanation on the first step of transforming the join ordering problem into a form suitable for quantum computing approaches, which involves the formulation of the input problem as a BILP problem. In addition, it was shown how the inequality constraints of the join ordering formulation can be turned into equality constraints via the introduction of slack variables, which serves as a precondition for the second step of the overall

transformation that will be described in this section.

Ising Model. Following [20], the formulation of a BILP problem with only equality constraints as an Ising Hamiltonian is given by $H = H_A + H_B$. The optimal solution to the problem is encoded by the ground state of the Hamiltonian H. Furthermore, the Sub-Hamiltonian H_A is given by

$$H_A = A \sum_{j=1}^{m} \left(b_j - \sum_{i=1}^{N} S_{ji} x_i \right)^2 . \tag{41}$$

H_A is formulated with the goal of invalidating each assignment of the binary variables x_i that leads to a violation of any of the m constraints in the BILP problem. S is a matrix of shape $m \times N$, which contains all coefficients for all N variables and m constraints, and $b \in \mathbb{R}^m$ is a vector containing the values of the right hand side of the constraining equations.

Any violating variable assignment means that the term $b_j - \sum_{i=1}^{N} S_{ji} x_i$ will not be zero and therefore, the energy penalty weight A will be added to the energy level of the Hamiltonian. As such, invalid assignments should not lead to a minimization of the overall energy. However, in order to achieve the desired effect, the value of A needs to be set specifically in relation to the Sub-Hamiltonian H_B, which is given by

$$H_B = B \sum_{i=1}^{N} c_i x_i. \tag{42}$$

H_A effectively encodes the constraints of the BILP problem. Similarly, H_B encodes its objective function. As such, $c \in \mathbb{R}^N$ is a vector containing the cost values of the objective function.

Note that in the problem formulation given in [20], H_B contains an additional negative factor since the problem is formulated as a maximization problem. By omitting this factor, the formulation becomes suitable for minimization problems like the join ordering problem, as both Sub-Hamiltonians need to add to the overall energy of the overarching Hamiltonian (albeit to varying degrees).

Penalty Weights. The task of determining suitable values for penalty weights such as A and B can typically be a tricky problem and greatly affects the quality of the solution. As it is desired to generate valid solutions rather than variable assignments that minimize cost while violating constraints, the relation $A \gg B$ needs to hold in the case of this particular QUBO formulation. At the same time, setting the penalty weights too high also has negative impacts on the solution quality. In quantum annealing, large penalty weights lead to a compression of the energy spectrum of the system and thus to a small minimum energy gap [41]. The result would be a large annealing time, which is inversely proportional to the minimum energy gap between the ground state and the first excited state of the Hamiltonian as explained in Section 3.5.

Further following [20], the violation of even a single constraint by the smallest possible value should add a sufficiently large energy penalty to the Hamiltonian so that such a variable assignment cannot correspond to its ground state. This is achieved by ensuring that any cost saving with regards to H_B that leads to a violation of a constraint is smaller than the energy penalty that is added through H_A as a result.

Postulating that the cost vector c only contains positive values (which is the case for the join ordering problem), an upper bound for the cost that can be saved regarding the objective function is given by

$$C = \sum_{i=1}^{N} c_i. \tag{43}$$

Assuming B is simply set to 1, C equals the maximum energy saving possible with regards to H_B. Thus, it needs to be ensured that the energy penalty for violating a single constraint by the smallest possible value is larger than C in order to remove any incentive to violate constraints.

In this work, the coefficient values in the matrix S and vector b are rounded to decimal places in accordance with the precision factor ω (which, as explained in Section 6.1.2, is needed for approximating continuous slack variables). The smallest possible value which any of the constraints that require continuous slack variables can be violated with is then ω, whereas for any other constraint the smallest value is 1 and thus equal or higher. As such, the smallest possible value a constraint can be violated with is given by ω.

The rounding of the coefficients is necessary due to the higher complexity of determining a suitable penalty weight A if the values are not rounded. In the latter case, it is not guaranteed that there exists an assignment for the slack variables so that $b_j - \sum_{i=1}^{N} S_{ji}x_i = 0$ even if the assignment of the logical variables is valid. As the upper bound for the resulting error depends on ω, it would be possible to specify a value for A that leads to an invalidation of any variable assignment that violates at least one constraint.

However, even for valid variable assignments, the possible errors for the affected constraints can accumulate and induce a high energy penalty, which should only be the case for an invalidation. In summary, enforcing that violating variable assignments are invalidated via sufficiently large penalty weights while also ensuring that valid assignments are not unreasonably penalized adds a lot of complexity to the problem of finding suitable penalty weights in case the coefficients are not rounded.

Resulting from the above, the following inequality must hold in order to enforce valid solutions:

$$A > \frac{C}{\omega^2}. \tag{44}$$

Squaring ω is necessary as the constraints are encoded within a quadratic term in H_A. It can be seen that, the lower the value for ω, the larger A has to be. Considering the negative effects

of large penalty weights on the quantum annealing process as explained above, the benefits of a higher precision through lower ω might be negated or even outweighed as a result.

Quadratic Contributions. It can be seen that H_A is the source for all quadratic contributions of the QUBO formulation. As such, all non-diagonal entries of the resulting QUBO matrix are caused by the specific constraints of the BILP problem, which are encoded by H_A. More specifically, a quadratic term for two variables x_i and x_j needs to be added to the QUBO matrix if those two variables appear together in at least one constraint.

As described in an earlier section, the more quadratic terms a QUBO matrix contains, the more complex the task of embedding the matrix on a quantum annealing machine becomes. The application of the QAOA algorithm on gate-based systems is likewise negatively impacted, as quadratic terms require more connections between the qubits, which are sparse on current machines. However, creating indirect connections via swap operations leads to higher circuit depths. More details on the emergence of quadratic terms in regards to the specific properties of the join ordering formulation will be explained in Section 6.3.3.

6.2 Implementation

The previous section explained a two-step approach for transforming the join ordering problem into QUBO form, which is suitable for the application of quantum computing. Based on that, this section describes the respective implementation for both the Qiskit and Ocean SDK. First, Section 6.2.1 gives an overview on different frameworks used for the implementation. Next, Section 6.2.2 explains the implementation details.

6.2.1 Frameworks Used

In this section, the frameworks which are used in the implementation will be explained. In order to deploy QUBO problems on D-Wave systems or to use local solvers, the D-Wave Ocean SDK can be used. A documentation for the SDK can be found in [42]. The SDK includes several packages which were used for this work, such as the dimod package (version 0.9.15), which contains classes representing quadratic programs such as the `BinaryQuadraticModel` class. `BinaryQuadraticModel` objects serve as an input for the solvers offered by the Ocean SDK. Additional solvers such as classical simulated annealing solvers are contained in the dwave-neal package (version 0.5.7). Finally, in order to conduct simulations for embeddings for the topology of real D-Wave systems, version 0.8.8 of the dwave_networkx package is used. This package provides tools for specifying objects representing Chimera or Pegasus topologies.

For the BILP formulation of the problem, the Gurobi Python interface gurobipy is used. The Gurobi optimizer can be used for solving mathematical optimization problems including MILP and BILP problems [43]. For this work, version 9.1.2 of gurobipy is used. Specific methods provided by gurobipy classes can be used for extracting the resulting coefficient

matrix as well as the coefficient and cost vectors, which serve as the input for the subsequent parts of the transformation.

Moreover, for the formulation of the Ising Hamiltonians, the PyQubo library (version 1.0.12) is used. PyQubo is a tool referenced in the D-Wave Ocean documentation that offers an intuitive way for creating QUBO models from mathematical expressions [44]. It additionally supports the generation of `BinaryQuadraticModel` objects, which serve as input models for the D-Wave Ocean SDK. Further library packages that are used in the implementation include the numpy package (version 1.21.0) as well as the math module which is included in the Python installation.

6.2.2 Implementation Details

This section will describe the details of the Python (version 3.8.8) implementation of the quantum computing approach for the join ordering problem. First, the input parameters for the implementation will be explained. As described in Section 4.2, the input to the join ordering problem consists of a query graph. More specifically, the input parameters include a list of cardinalities referring to their respective relations that are to be joined, as well as an adjacency list containing the edges of the graph, which represent the join predicates with each predicate referring to two relations.

Moreover, the input parameters include a list of predicate selectivities corresponding to the join predicates and a list of threshold values, which are used for approximating the actual non-logarithmic cardinalities of the intermediate results as described in Section 6.1.2. Finally, the input contains a parameter for specifying the number of decimal positions p which determines the precision factor ω for the approximated continuous slack variables.

The following shows the overall steps included in the implementation:

1. Calculate the logarithmic values from the input cardinalities and threshold values.
2. Determine which variables can be pruned.
3. Create the Gurobi `Model` object and add all necessary variables to it.
4. Define the objective function for the `Model`.
5. Add all needed constraints and add slack variables where necessary.
6. Extract the coefficient matrix as well as both the coefficient vector and the cost vector.
7. Calculate the penalty weight for the invalidating Ising Hamiltonian.
8. Formulate the Ising Hamiltonian.

In the first step, the logarithmic values of the relation cardinalities and the threshold values are calculated, as they will be necessary for the formulation of the BILP constraints. Afterwards,

depending on the cardinalities and threshold values, it is determined which variables are actually needed and which may be pruned. As will be explained in Section 6.3.1, a variable cto_{rj} is only needed if the r-th threshold value can be reached in join j.

Next, the GUROBI `Model` object is initialized and all variables that need to be present are added to it. Note that a variable cto_{rj} is only included for $j > 0$, as the objective is to minimize intermediate relation cardinalities, which equal the cardinalities for the outer join operands. However, the outer operand for the very first join is merely one of the input cardinalities. As such, all cto_{r0} variables can be omitted, which reduces the number of required qubits. Likewise, all pao_{p0} variables are omitted, as the usage of a predicate requires the two relations it refers to to be included in the outer operand of the respective join. However, for the very first join, the outer operand consists of only one relation, which makes pao_{p0} variables superfluous. As such, they are pruned in order not to unnecessarily waste qubits.

Furthermore, the objective function as given in Equation 38 is defined for the `Model` and all necessary constraints are added as specified in Section 6.1.2. For the transformation of inequality constraints to equality constraints, additional slack variables need to be added to the `Model`. As mentioned in Section 6.1.3, only one additional slack variable is needed for the majority of constraints whereas in most cases, several binary slack variables are required for the approximation of the continuous slack variables needed for constraints of type 7. Specific details about the required number of additional slack variables are explained in Section 6.3.1.

The next step after completing the creation of the GUROBI `Model` involves formulating the Ising Hamiltonian. For this, the coefficient matrix and both the coefficient vector and cost vector need to be extracted from the `Model`. Fortunately, the `Model` class offers methods to extract them either directly or to extract data which the matrix and vectors can be easily derived from. As a last step before formulating the Ising Hamiltonian, the weight for the invalidating Sub-Hamiltonian needs to be calculated based on the cost vector and precision factor as explained in Section 6.1.4.

Then, with the goal of using the D-Wave OCEAN SDK, the PYQUBO tool can be utilized to formulate the Ising Hamiltonian using mathematical expressions. After compiling the PYQUBO `Model`, it can be directly transformed into an instance of the `BinaryQuadraticModel` class via the `to_bqm()` method. As such, a direct input for the OCEAN solvers can be generated using the PYQUBO tool. Alternatively, a DOCPLEX `Model` object may be created in the same manner, which, as shown in previous sections, can be used to create an instance of the `QuadraticProgram` class. The latter serves as an input for the solvers of the QISKIT SDK, which are initialized in the same manner as described for the MQO simulations in Section 5.2.2.

The `BinaryQuadraticModel` object serves as an input for solvers such as the classical simulated annealing solver of the DWAVE-NEAL package. However, since the goal is to conduct simulations for the Pegasus topology of the D-Wave Advantage system, the classical solver object is not directly used but instead passed as an input parameter for creating an object of

the DIMOD `StructureComposite` class. Moreover, objects representing the nodes and edges of a Pegasus P16 topology are created with the DWAVE_NETWORKX package and additionally passed as input parameters. The `StructureComposite` object can then be used for solving a problem represented by a `BinaryQuadraticModel` object based on a real topology. Finally, the object is moreover used as an input parameter for creating an object of the Ocean class `EmbeddingComposite`, which automatically finds an embedding for the `BinaryQuadraticModel` for the topology of the `StructureComposite` based on the minorminer algorithm.

6.3 Evaluation

In the previous sections, an overview on the two-step transformation of the join ordering problem into a form suitable for quantum computing was given and details on the implementation were described. This section will evaluate the approach and present simulation results. First, Section 6.3.1 will explain upper bounds on the number of required variables for a join ordering problem. Section 6.3.2 will show the scaling behavior for the required number of logical qubits with regards to the dimension of the join ordering problem. Next, Section 6.3.3 analyzes how different problem parameters influence the application of quantum computing approaches. Section 6.3.4 will present simulation results for IBM-Q systems and finally, Section 6.3.5 will likewise show simulated results for D-Wave systems.

6.3.1 Upper Bounds for the Number of Binary Variables

Section 6.1.2 gave an overview on all types of binary variables which are needed to formulate the join ordering problem as an MILP problem. This includes the logical variables expressing properties directly related to the join ordering problem (e.g., variables denoting whether or not a relation is part of the inner or outer operand of some join). However, in order to further bring the problem into BILP form that can be transformed into QUBO form, additional slack variables have to be added to the problem formulation in order to eliminate inequality constraints as described in Section 6.1.3.

Taking all of these variable types into account, this section will explain the upper bounds for the number of variables needed for the complete problem formulation. The upper bounds depend on problem specific parameters like the input cardinalities and the precision of the approximated continuous slack variables. Since one qubit represents one binary variable, the number of required qubits is equal to the total number of binary variables. As such, the following will illustrate the qubit requirements by showing how the number of required binary variables for formulating the join ordering problem in the manner described in previous sections depends on the specific parameters of the join ordering problem.

Let n be the total number of binary variables needed to encode the join ordering problem as a

BILP problem. Then, n is given by

$$n = n_{\text{log}} + n_{\text{bsl}} + n_{\text{csl}}, \tag{45}$$

where n_{log} denotes the number of logical variables consisting of all variables tii_{tj}, tio_{tj}, pao_{pj} and cto_{rj}. The semantics of these variables have been explained in Section 6.1.2. Due to the aforementioned pruning of all unnecessary cto_{rj} variables, an upper bound for n_{log} is given by

$$n_{\text{log}} \leq T \cdot J + T \cdot J + P \cdot (J-1) + R \cdot (J-1) = J(2T + P + R) - P - R, \tag{46}$$

where T denotes the number of relations to be joined, $J = T - 1$ is the number of joins, P denotes the number of predicates and R is the number of threshold values. Note that the calculation takes into account that pao_{p0} and cto_{r0} variables are omitted due to the reasons explained in Section 6.2.2.

The term n_{bsl} denotes the number of slack variables added for all inequality constraints for which a single slack variable suffices to transform them into equality constraints. This includes all constraints of type 3, 5 and 6. Therefore, n_{bsl} is given by

$$n_{\text{bsl}} = J \cdot T + P \cdot (J-1) + P \cdot (J-1) = J(T + 2P) - 2P. \tag{47}$$

Once again, the equation takes into account that pao_{p0} variables are omitted.

Finally, n_{csl} denotes the number of binary variables needed to express approximated values for all continuous slack variables required for constraints of type 7. In order to determine n_{csl}, an upper bound for the continuous slack variables that need to be approximated needs to be determined as a prerequisite.

Following from Equation 39, for any threshold value r and join j the upper bound C_{rj} for the continuous slack variable sl_{rj} is given by

$$C_{rj} = \log(\theta_r) + \infty_{rj} \geq \log(\theta_r) + cto_{rj}\infty_{rj} - lco_j = sl_{rj}, \tag{48}$$

where lco_j (as it is referred to in [16]) is the logarithmic cardinality of the outer operand for join j:

$$lco_j = \sum_t \log(Card(t)) tio_{tj} + \sum_p \log(Sel(p)) pao_{pj}. \tag{49}$$

It can be seen that the upper bound for sl_{rj} depends on the term ∞_{rj}, which is a sufficiently large constant that fulfills a type 7 constraint in case lco_j exceeds $\log(\theta_r)$ as cto_{rj} is set to 1. Since the upper bound should be kept as low as possible for lower qubit requirements, the minimum value that suffices in all cases when $cto_{rj} = 1$ is chosen for ∞_{rj} and needs to be determined next. Let the term mlc_j thereby denote the maximum logarithmic cardinality the

outer operand of the j-th join may have. Specifically, mlc_j is given by

$$mlc_j = \sum_{t=1}^{j} \log(Card(t)), \tag{50}$$

where $Card(i) \geq Card(j) \forall i < j$. We consider only those cases where $mlc_j > \log(\theta_r)$, since otherwise, the threshold value is never reached and as a result, variable cto_{rj} is never activated. As such, neither the variable nor the constraint are even necessary in those cases and can be pruned, which also diminishes the necessity for the approximated continuous slack variable. Therefore, the following will determine an upper bound for n_{csl} for the worst case in which none of the variables can be pruned.

Following from constraint type 7, for any join j and threshold value r, the lower bound for ∞_{rj} when $cto_{rj} = 1$ is then given as

$$\infty_{rj} \geq mlc_j - \log(\theta_r) \geq lco_j - \log(\theta_r). \tag{51}$$

Setting ∞_{rj} to its lower bound and inserting it into Equation 48 produces $C_{rj} = mlc_j$. Therefore, the number of binary slack variables n_{bin} necessary for the discretization of sl_{rj} as shown in Equation 40 is given by

$$n_{\text{bin}} = \left\lfloor \log_2 \left(\frac{mlc_j}{\omega} \right) \right\rfloor + 1. \tag{52}$$

Following Equation 52, which expresses the number of additional binary variables for a single constraint, an upper bound for the total number of required binary variables n_{csl} for all type 7 constraints is given by

$$n_{\text{csl}} \leq R \sum_{j=2}^{J} \left(\left\lfloor \log_2 \left(\frac{mlc_j}{\omega} \right) \right\rfloor + 1 \right). \tag{53}$$

Like before, only the constraints for the corresponding cto_{rj} variables for $j > 1$ are accounted for. Finally, following from Equation 45, Equation 46, Equation 47 and Equation 53, the upper bound for the overall number of binary variables is given by

$$n \leq 3TJ + (J-1)(3P+R) + R \sum_{j=2}^{J} \left(\left\lfloor \log_2 \left(\frac{c_j}{\omega} \right) \right\rfloor + 1 \right). \tag{54}$$

It can be noted that the number of necessary qubits is at least quadratic in the number of relations to be joined (as $J = T - 1$). In general, considering the above equations, it is clear that the number of required logical qubits is highly dependant on the number of joins in the join ordering problem.

6.3.2 Logical Qubit Scaling Behavior

This section will give an overview on the scaling behavior of the number of required logical qubits as the dimension of a join ordering problem increases. More specifically, it will first show how the number of qubits increases with the number of relations in the input of the join ordering problem while also considering the number of predicates. In addition, the scaling behavior for an increasing number of threshold values and precision factors will also be described.

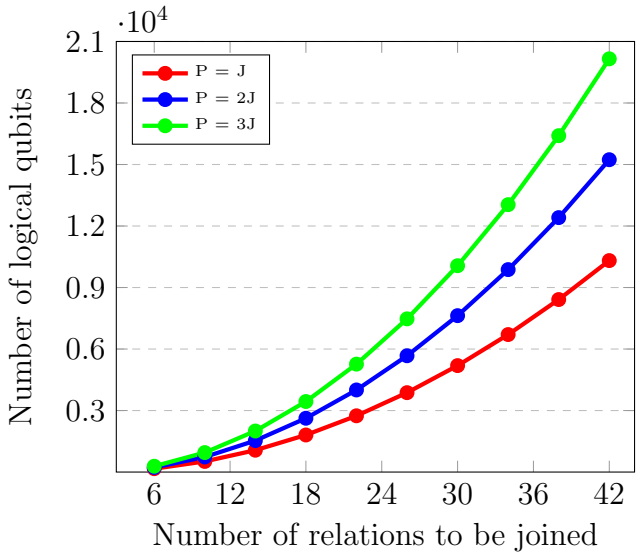

Figure 11: The qubit scaling behavior for the join ordering problem with respect to the number of relations T and varying numbers of predicates P. Specifically, the chart shows graphs for $P = J$, $P = 2J$ and $P = 3J$ where $J = T - 1$ denotes the number of joins.

Influence of the Number of Relations and Predicates. Figure 11 illustrates the scaling behavior of the number of required qubits, which is denoted by the y-axis, in relation to the number of relations in the join ordering problem, which is denoted by the x-axis. It should be noted that Figure 11 refers to the number of logical qubits. As explained in earlier sections, the actual number of qubits that quantum computing systems need to offer can be higher than these numbers, which is typically the case for D-Wave systems. As such, the qubit number shown here should be seen as a lower bound for a join ordering problem with regards to the parameters which are explained below.

Moreover, Figure 11 shows a comparison between different predicate numbers P, specifically for $P = J$, $P = 2J$ and $P = 3J$ where J is the number of joins. The remaining problem parameters have been set in such a way that they only have a negligible impact on the number of qubits. Specifically, the actual cardinality is approximated via only 1 threshold value. Furthermore, each relation has a cardinality of 10 and the precision factor for the approximation of the continuous slack variables is set to 1. Finally, in order to represent a more general join ordering problem, Figure 11 does not take pruning of *cto* variables into account and thus depicts the upper bounds for the number of necessary logical qubits.

Figure 11 shows the qubit requirements for problem instances with up to 42 relations. The largest problems that were solved with a classical MILP solver in [16] contained 60 relations as an input. Looking at the required qubit number for a join ordering problem with 42 relations if $P = J$, which is approximately 10,000 qubits, it becomes clear that neither current D-Wave machines nor current IBM-Q quantum systems offer enough qubits to solve problems of dimensions comparable to the larger ones solved in [16]. Evidently, the same is true for even higher predicate numbers featured in Figure 11.

Figure 11 also illustrates the high impact of the number of predicates P on the overall number of logical qubits. For 42 relations, doubling the number of predicates roughly leads to a 50% increase in the number of qubits. Moreover, the graph for $P = J$ represents the lower bound for the problem dimension of a practical join ordering problem with regards to the predicate number. Even fewer predicates would necessitate the usage of cartesian products, which are not even considered by some query optimizers.

Influence of an Increasing Precision. This lower bound does so far not take the remaining problem parameters, specifically the number of threshold values and the precision factor ω, into account. For a practical join ordering problem, these parameters are important for determining meaningful results and also influence the overall number of logical qubits as they impact the number of *cto* variables as well as the upper bound for the number of binary variables needed to approximate the continuous ones. This upper bound also depends on the cardinalities of the relations which are to be joined. However, these only have a minor impact on the total qubit number compared to other parameters even when dealing with relations of large dimensions, as the worst-case intermediate cardinality mlc_j for join j equals the accumulated logarithmic input cardinalities and not the actual cardinalities.

Figure 12 illustrates the impact of tuning the approximation quality of the actual resulting cardinality for increasing numbers of up to 20 threshold values, which are denoted by the x-axis. Once again, the y-axis denotes the number of logical qubits. In addition, Figure 12 includes graphs for varying precision factors, specifically for $\omega = 1$, $\omega = 0.01$ and $\omega = 0.0001$. The remaining parameters are set as follows: the number of relations to be joined is 20 and the number of join predicates is 19, equaling the number of joins. All relations have a cardinality of 10. Finally, pruning is not accounted for, again with the goal to represent a more general join ordering problem.

As the input of a join ordering problem contains a significant number of joins and join predicates, the qubit number is already at a high level for 2 threshold values. It can be seen that, even for $\omega = 1$, an increase in the number of threshold values, which means more precision for the approximation of the actual intermediate cardinalities, leads to a significant overhead in required qubits. For $\omega = 0.01$, increasing the number of threshold values from 2 to 14 approximately leads to a 94% increase in the number of qubits. However, the overhead in qubits becomes even more drastic for lower ω. This can be seen when comparing the qubit numbers

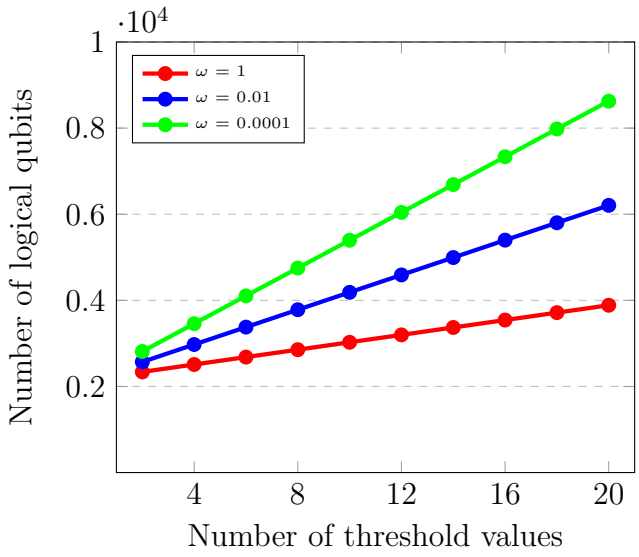

Figure 12: The qubit scaling behavior for the join ordering problem with respect to varying approximation factors ω. Specifically, the scaling behaviors for $\omega = 1$, $\omega = 0.01$ and $\omega = 0.0001$ are compared.

for the join ordering problem with 20 threshold values for $\omega = 1$ and $\omega = 0.0001$. For the former, approximately 4,000 qubits are necessary, whereas for the latter, over 8,000 qubits and thus more than twice as many are required.

It can be seen that the impact of a high number of threshold values and low values for the parameter ω on the required qubit number is quite large. As stated above, even if these parameters are neglected, as it was done in Figure 11, the number of qubits required to solve practical join ordering problems exceeds the qubit numbers offered by current quantum computing systems by a large margin. In contrast to the variables needed for the remaining problem parameters, which ensure that a solution is valid, the additional variables for the number of threshold values and ω merely influence the quality of the solution. As such, they can be freely tuned, which should be done in a careful manner as to not unnecessarily waste qubit resources while still enabling a sufficient solution quality.

6.3.3 Parameters Affecting the Number of Quadratic QUBO Terms

The previous section gave an overview on the required number of logical qubits in order to solve join ordering problem instances of varying dimensions on quantum computing machines. However, the number of qubits for a problem formulation is not the only relevant factor that influences the applicability of quantum computing. As already explained in earlier sections, the number of quadratic terms within a QUBO matrix also plays a major role for both the applicability of the QAOA algorithm on gate-based systems and quantum annealing. For the former, a higher number of quadratic terms leads to longer circuits, making them more error-prone, whereas for the latter, the process of finding qubit embeddings becomes more complex.

As already briefly explained in Section 6.1.4, the number of quadratic terms depends on the structure of the BILP problem. Specifically, one quadratic term is necessary for every pair of variables that appears in at least one constraint. In this section, the effect of different problem parameters on the number of emerging quadratic terms is analyzed. The analysis also includes the resulting circuit depth for the QAOA algorithm. However, only problem instances of smaller dimensions can be considered, as the number of qubits available on IBM-Q quantum devices and general simulators like the qasm simulator is very limited. Specifically, problem instances with more than three relations to be joined already exceed these limits, so all three of the compared join ordering problems will be limited to three relations.

Table 4: A comparison of different join ordering problem instances with regards to their input parameters and resulting number of quadratic terms and circuit depths.

	Problem 1	**Problem 2**	**Problem 3**
Number of predicates	3	0	0
Number of threshold values	1	4	1
Precision factor ω	1	1	0.001
Required logical qubits	30	30	30
Resulting number of quadratic terms	70	84	138
Resulting circuit depth (QAOA)	63	72	99

Table 4 shows the three problem instances with their respective parameter values as well as the resulting number of logical qubits and the resulting number of quadratic terms. In addition, Table 4 includes a row showing the depth of the QAOA circuit. Since it does not play a role for this comparison whether or not the problems have one unique solution, all relations share the same cardinality, specifically 10. Once again, pruning is not accounted for. As such, only the number of threshold values is relevant, not their concrete values. Moreover, all problems produce the same qubit requirements, specifically 30 logical qubits.

However, these qubit requirements have been caused by different parameters for each different problem. The relevant parameters whose effects have been analyzed are the number of predicates, the number of threshold values and finally, the precision factor ω. For each problem, one of these parameters has been set in such a way that leads to an increase in the number of qubits until 30 qubits are required, while the remaining two parameters are set in a way that leads to otherwise negligible qubit increases. More specifically, for problem 1, the number of predicates has been increased to 3, for problem 2, the number of threshold values has been increased to 4, and finally, for problem 3, ω has been set to 0.001.

It can be seen that the different parameter settings lead to varying numbers of resulting quadratic terms. The difference is particularly striking when comparing problem 1, which requires 70 quadratic terms, to problem 3, which needs 138 quadratic terms and thus roughly twice as many. These differences can be explained when considering the effects of the pa-

rameter adjustments on the structure of the resulting BILP problem, or more specifically, the constraints.

Looking at problem 1, an increase in the number of predicates by one leads to one additional variable of type *pao* for every join excluding the first one. With only two joins in total, this means one additional variable per predicate is added. In addition, for each new *pao* variable, two more constraints, specifically one for type 5 and one for type 6, need to be added (details for the constraint types have been explained in Section 6.1.2). Each constraint of type 5 and 6 contains exactly 3 binary variables (including one slack variable), so 6 new quadratic terms are added for each new predicate due to those two constraints.

Moreover, assuming no pruning is applied, each new variable of type *pao* appears in at least one constraint of type 7. These constraints are particularly critical with regards to the number of quadratic terms as they include significantly more variables than those of other types. Specifically, adding an additional *pao* variable to such a constraint leads to one additional quadratic term for every relation, every already existing predicate and for the *cto* variable of the respective constraint. Furthermore, an additional quadratic term is necessary for every binary slack variable that is used for approximating the continuous slack variable.

In contrast to problem 1, the parameters which are adjusted for problems 2 and 3 solely influence constraints of type 7. For problem 2, one additional constraint of type 7 is added for each additional threshold value as the problem includes only one relevant join. Due to the low number of relevant joins, the increase in type 7 constraints and thus the increase in quadratic terms for problem instance 2 is not that significant. In general, however, the number of threshold values is a big impact factor on the number of quadratic terms, as the impact increases further with each additional join.

Adding a type 7 constraint does, in general, not mean that new quadratic terms need to be added for every pair of variables within those constraints, as many of these pairs already appear in prior type 7 constraints and are therefore already accounted for. Moreover, for problem 2, the number of variables within each of the additional type 7 constraints is still lower than the number of variables contained in the single type 7 constraint in problem 3. On the other hand, decreasing ω has drastic effects on the number of quadratic terms, as this parameter adjustment leads to an increase of binary slack variables used for approximating the continuous slack variables in every single constraint of type 7. The effect is drastic even for a low number of joins.

In summary, especially the number of threshold values and the precision factor ω significantly impact the number of quadratic terms. The purpose of lower ω is to increase the precision of the solutions. However, this comes at the cost of a significant increase in quadratic terms. It is thus possible that the benefits of lower values for ω are to some degree diminished or even outweighed by the negative impact of the resulting higher number of quadratic terms on both gate-based and annealing based quantum computing approaches.

6.3.4 Evaluation for IBM-Q Systems

This section will evaluate the applicability of the hybrid quantum algorithms described in an earlier section on an IBM-Q device for the join ordering problem. This is done in a similar manner as in the previous chapter, which analyzed the applicability of IBM-Q systems for MQO. Once again, the depths of the quantum circuits resulting from the application of the hybrid algorithms are analyzed.

The simulations have been conducted using the remote qasm simulator provided by IBM-Q. The qasm simulator is a general simulator that offers 32 qubits. Information about the specifications of IBM-Q simulators and real quantum systems can be found in [3]. As described in the previous section, this sets strict limits on the dimensions of the join ordering problem for which simulations are possible. As such, only very small problem instances with merely 3 relations can be simulated. This circumstance already partially answers the question to which extent current gate-based quantum devices can be used. However, analyzing the circuit depths for such smaller join ordering problems provides additional insights.

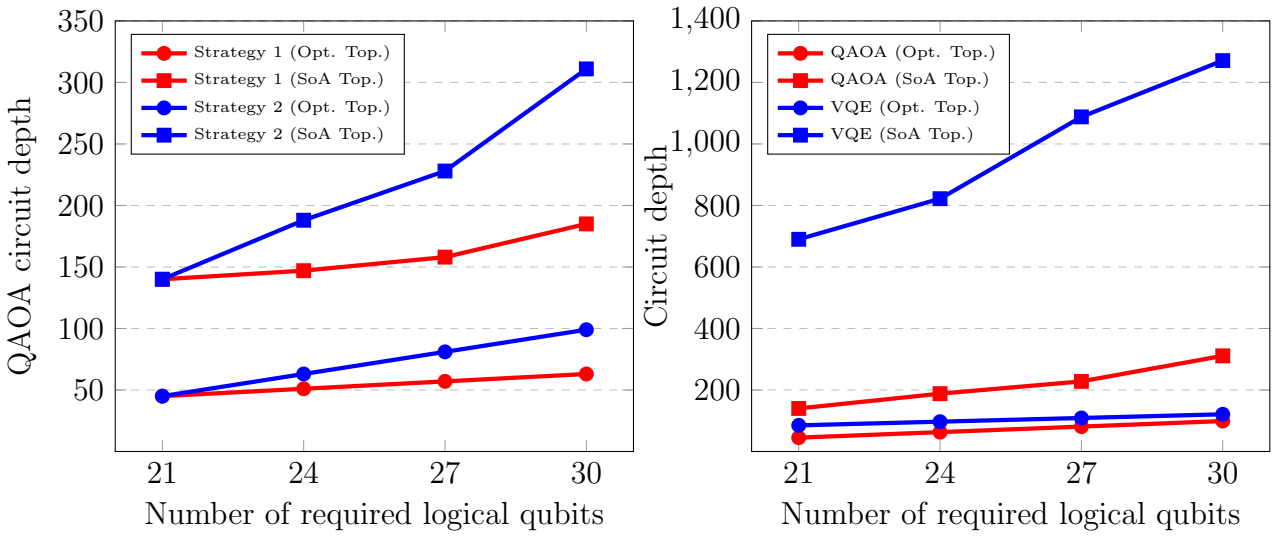

Figure 13: The scaling behavior of the circuit depth in relation to the number of qubits, the applied algorithms and the properties of the join ordering problem.

Circuit Depth Analysis. Figure 13 depicts two plots which show the scaling behavior of the circuit depths in relation to various properties. Once again, the resulting circuit depths for the two algorithms QAOA and VQE have been analyzed. Moreover, similarly to the experiments for MQO, both the optimal qubit topology and a state-of-the-art topology have been used for the simulations. For the latter, the charts depict the average circuit depths for 20 samples resulting from the heuristic transpilation process.

Moreover, different from the simulations for MQO, the topology for the IBM-Q Brooklyn system was used for these simulations instead of the Mumbai topology. This was necessary because

the qubit requirements for some of the simulations exceeded the number of qubits offered by the Mumbai machine, which only provides 27 qubits. In contrast, the Brooklyn device offers 65 qubits. In turn, its calibrated coherence times are lower than those of the Mumbai system. The average coherence times for the Brooklyn machine at the time of writing are 66.02 µs for T1 and 79.44 µs for T2 and thus significantly lower than the ones for the Mumbai device (115.83 µs for T1 and 104.86 µs for T2).

Finally, the QAOA circuit depth depends not only on the number of qubits, but also on the number of required connections between qubits and thus on the number of quadratic terms in the QUBO matrix. Therefore, join ordering problems have been created using two different strategies which influence the number of quadratic terms. More specifically, for strategy 1, join ordering problems with higher qubit requirements are created by increasing the number of predicates, whereas for strategy 2, the same is achieved by decreasing the precision factor ω instead. The latter strategy leads to significantly higher numbers of quadratic terms as explained in Section 6.3.3. In all cases, the join ordering problem is applied on an input with three relations, each containing 10 tuples, and only one threshold value.

Both charts depicted in Figure 13 show the circuit depths, which are denoted by the y-axis, for increasing numbers of qubits, which are given by the x-axis. The left chart exclusively depicts circuit depths resulting from QAOA and includes graphs for the two different strategies described above. It also features graphs for both the optimal topology as well as the Brooklyn topology. The right chart also shows graphs for the different topologies, however, instead of different strategies it includes graphs for both QAOA and VQE. For the right chart, all circuit depths for QAOA are derived using strategy 2. For VQE, the applied strategy has no influence on the circuit depths.

Again, similarly to the MQO observations, looking at both charts it becomes apparent that transpiling the circuits based on a state-of-the-art topology leads to a significant increase in the circuit depth. This is, once again, especially true when looking at the graph for VQE and the Brooklyn topology in the right chart. In addition, the negative impact of an increase in quadratic terms can be observed in the left chart. While the overhead in circuit depth caused by strategy 2 is small for low qubit numbers, it significantly increases as the number of required qubits gets larger and is at roughly 57% for 30 qubits. However, this overhead increases even more and is at roughly 68% when considering the results for the Brooklyn topology.

Circuit Depth and Coherence Times. Once again, as it was done for the evaluation of MQO, calculating a threshold for the maximum circuit depth that can be reliably executed on the state-of-the-art quantum device is useful for evaluating these circuit depths. It can be calculated based on the lowest coherence time for the device and the average gate time. For the Brooklyn device, the coherence times $T1_{\text{Br}}$ and $T2_{\text{Br}}$ are as stated above whereas the average gate time is given by $g_{\text{avg}} = 370.469$ ns. Based on these values, the threshold for the circuit

depth d_{Br} is given by

$$d_{\text{Br}} = \left\lfloor \frac{\min(T1_{\text{Br}}, T2_{\text{Br}})}{g_{\text{avg}}} \right\rfloor = \left\lfloor \frac{66{,}020 \text{ ns}}{370.469 \text{ ns}} \right\rfloor = 178. \tag{55}$$

It can be noted that the threshold circuit depth for the Brooklyn system is approximately 28% smaller than the one for the Mumbai system, which was used for the MQO simulations and for which a threshold depth of 248 was calculated. As mentioned above, this is due to worse properties such as lower coherence times which are likely caused by offering more qubits.

It becomes clear that applying VQE for reliably solving join ordering problems on the IBM-Q Brooklyn machine or devices with similar properties is infeasible, as all resulting circuit depths exceed the threshold depth by a large margin. On the other hand, using QAOA for solving join ordering problems which have been created using strategy 1 can be reliably done until around 30 qubits (the depicted average circuit depth for 30 qubits and the Brooklyn topology very slightly exceeds the maximum depth, however, the transpilation algorithm may sometimes still output smaller circuits).

However, when considering the Brooklyn circuit depths for strategy 2, starting from 24 qubits, all of them exceed the threshold depth, which further underlines the negative impact of quadratic terms on the applicability of current quantum devices. On the other hand, assuming the transpilation process for the Mumbai topology produces similar or even smaller circuit depths, a join ordering problem with 24 qubits produced by strategy 2 can still be executed within the coherence time of the Mumbai system, as its circuit depth is slightly below the calculated threshold for the Mumbai system. It can also be observed that, even for strategy 1, join ordering problems with qubit requirements of more than 30 qubits cannot reliably be solved on the Brooklyn machine.

6.3.5 Evaluation for D-Wave Systems

This section will be focused on evaluating the applicability of quantum annealing systems on the quantum computing approach for the join ordering problem. In comparison to the gate-based devices, D-Wave machines offer significantly more physical qubits. The currently largest device in terms of physical qubits offered by D-Wave is the Advantage system, which features over 5,000 qubits [8]. This number exceeds the number of qubits offered by current IBM-Q machines by orders of magnitude. However, typically not all physical qubits of annealing machines can represent one logical qubit of a problem as opposed to the gate-based quantum machines.

Instead, several physical qubits form a chain to represent one logical qubit. The length of these chains depends on the specific properties of the problem. More details about two different D-Wave topologies have been explained in Section 3.6.2. Out of these, the Pegasus topology offers a superior qubit connectivity and is used for the D-Wave Advantage system. More specifically, the machine has a P16 topology [8]. For the following experiments, the `EmbeddingComposite`

class included in the D-Wave Ocean SDK is used to heuristically find minor embeddings for the P16 topology. Internally, this class uses the heuristic minorminer algorithm.

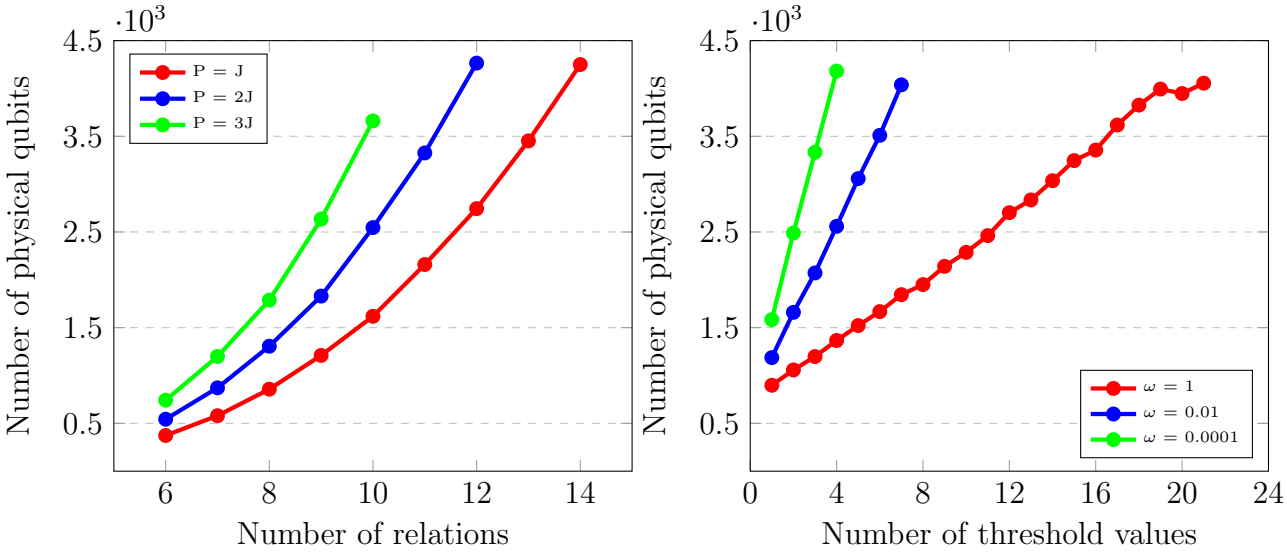

Figure 14: The scaling behavior of the number of required physical qubits in relation to the dimensions of the join ordering problem.

Figure 14 shows the results of the experiments. In both charts, the y-axis denotes the average number of required physical qubits based on 20 heuristically found embeddings for join ordering problems with varying parameters. The charts include only those problem dimensions for which an embedding can be reliably found, excluding those for which an embedding can only be determined in less than 50% of the cases. Once again, no pruning of *cto* variables is taken into account for both charts.

In the left chart, the x-axis denotes the number of relations in the input of the join ordering problem, ranging from 6 relations to a maximum of 14 relations. Moreover, it shows different graphs for varying predicate numbers P with respect to the number of joins J. The remaining parameters, specifically the number of threshold values and the precision factor ω, have both been set to 1 and thus have a negligible impact on the number of required qubits. Since pruning is not accounted for, the specific threshold values do not play a role.

As explained in an earlier section, the graph for $P = J$ can be seen as a lower bound regarding practical problem dimensions, as for even fewer predicates, cartesian products necessarily need to be taken into account, which are not even considered by some query optimizers. As such, the maximum number of relations a join ordering problem may take into account so that an embedding can still be reliably found for the D-Wave Advantage system is 14 (again, considering more general problems by not taking into account pruning). As the number of predicates increases, the maximum number of relations for which an embedding can reliably be found decreases. Notably, the maximum problem size for $P = 3J$ is given by 10 relations in the problem input and is therefore roughly 29% smaller than the one for $P = J$ with respect to

D-Wave Advantage system.

However, these results assume a negligible number of threshold values and ω, which significantly impact the number of quadratic terms as explained in Section 6.3.3, making the task of finding an embedding even more difficult. The concrete impact of these parameters on the feasibility of finding embeddings can be seen in the right chart. More specifically, it shows the impact of an increasing number of threshold values on the number of required physical qubits for a join ordering problem with 8 relations and $P = J$. Furthermore, it includes graphs for varying ω.

It can be seen that increasing the number of threshold values to improve the approximation accuracy leads to a significant overhead in the number of required physical qubits. This overhead becomes even larger for decreasing ω. For $\omega = 1$, increasing the number of threshold values from 1 to 7 increases the average number of physical qubits from 898 to 1845 and therefore by roughly 105%. However, for $\omega = 0.01$, merely adding 3 threshold values already increases the average number of required qubits by around 116%. In the extreme scenario of $\omega = 0.0001$, the maximum number of threshold values for a join ordering problem that considers 8 relations and $P = J$ in order to still reliably find embeddings is 4, whereas the problem may consider up to 21 threshold values if $\omega = 1$.

Summary. In summary, it becomes clear that the D-Wave Advantage system can be used to solve join ordering problems of a much higher dimension than what is possible on IBM-Q systems. However, there is still a big gap between the dimensions of these join ordering problems and the ones solved by a classical MILP solver as described in [16], which include queries with up to 60 relations. For the D-Wave Advantage system, the highest number of relations practical queries (with $P = J$ as the minimum number of predicates) may include when heuristically searching for an embedding is 14. However, also taking into account a non-negligible number of threshold values further increases the difficulty of reliably finding embeddings, especially when specifying lower values for ω.

7 Discussion

The first goal of this work was to examine the applicability of the current gate-based IBM-Q quantum systems on the MQO problem in comparison to the results of the quantum annealing approach presented in [9]. It was found that the MQO problem dimensions which can presently be solved using IBM-Q machines are considerably more limited when compared to all MQO problem classes which were solvable on the quantum annealing device used in [9]. The main limiting factor regarding the applicability of the quantum systems is the number of qubits, since one logical qubit is required to represent one plan in the MQO problem input. Despite the fact that several physical qubits offered by D-Wave systems are needed to represent one plan, the number of remaining logical qubits still far outweighs the number of qubits featured on current IBM-Q systems.

However, further limitations with respect to the specific MQO problem, in addition to the required number of qubits, need to be considered for IBM-Q systems. More specifically, the depth of the quantum circuit when using the QAOA algorithm depends on the number of quadratic terms in the QUBO matrix and as such, on the number of alternative PPQ and the number of possible cost savings. As the circuit depth increases, the occurrence of decoherence errors becomes increasingly more likely.

The results after analyzing the QAOA circuit depths for the qubit topology of the Mumbai system for different MQO problem classes made it clear that a reliable circuit execution without decoherence errors is unlikely for specific MQO problem classes such as ones with 3 queries and 8 alternative PPQ. In contrast, the circuit depth for the VQE algorithm is not dependant on the number of quadratic terms. However, for VQE, the results showed that the circuit depths for MQO problems with more than 12 plans far exceed the coherence times for the Mumbai system.

Summarizing the results for this research question, it is possible to utilize current IBM-Q systems for small-scale MQO problems. However, these systems can currently not be used to solve problems in similar dimensions to any of the problem classes discussed in [9]. This might, however, change in the future, when the number of qubits offered by IBM-Q systems increases, especially considering that one qubit is sufficient to represent one logical variable, which is not the case for current D-Wave systems where several physical qubits are required. Moreover, future IBM-Q systems also need to improve with respect to the coherence time so that the QAOA algorithm can reliably be applied on more complex MQO problems, which lead to more quadratic terms in the QUBO matrix.

The second goal of this work was to answer the research question of how the join ordering problem can also be solved on current quantum systems and to which extent. Moreover, the gate-based systems by IBM-Q were to be compared to the D-Wave quantum annealing machines in terms of applicability. It was shown that it is indeed possible to bring the problem into a

form which is suitable for current quantum machines. When compared to MQO, the approach for solving the join ordering problem on quantum systems which was presented in this work requires more reformulation steps. Instead of a direct reformulation of the problem, it was shown how the problem can first be transformed into an MILP problem using the approach described in [16]. It is then furthermore brought into BILP form by eliminating inequality constraints and by discretizing any continuous variable depending on an arbitrary precision factor. Following [20], the BILP problem can then be reformulated as a QUBO problem.

After conducting simulations for both current IBM-Q systems based on the Brooklyn qubit topology as well as the topology of the D-Wave Advantage system, it was found that once again, significantly larger problem dimensions can be solved on the quantum annealing machine compared to the gate-based one. For the former, the minorminer tool was able to heuristically determine embeddings for inputs of practical join ordering problems with up to 14 relations. However, this number should be considered an upper bound, since it only considers negligible values for both the number of threshold values and the precision factor. Both of these parameters influence the quality of the solution.

Compared to the D-Wave Advantage system, the problem dimensions which could be solved on current IBM-Q devices are very limited due to the number of qubits offered on these systems. More specifically, only very small join ordering problems with merely three relations to be joined could be simulated using a general simulator which supports only up to 32 qubits. However, even when considering real systems with up to 65 qubits such as the IBM-Q Brooklyn system, only slightly larger problems may become solvable due to several circumstances.

For one, the steep scaling behavior of the number of required qubits with respect to increasing problem dimensions, which was pointed out in Section 6.3.2, leads to high qubit requirements even for small problems. Moreover, depending on the input parameters of the specific problem, the likeliness of decoherence errors once again imposes further restrictions. This has been highlighted by the fact that some of the investigated join ordering problems, which can be solved with 32 qubits, already lead to circuits which cannot be executed within the coherence time of the Brooklyn system, thus making decoherence errors likely.

The number of threshold values and the precision factor have a particularly negative impact on the number of quadratic terms in the QUBO matrix and thus on the QAOA circuit depth. The circuits for problems with a higher number of predicates but fewer threshold values and a less influential precision factor were mostly found to be executable within the coherence time of the Brooklyn system. In contrast, the circuit execution time for problems requiring 24 qubits or more which have more threshold values and smaller precision factors exceeds the coherence time. Finally, the results showed that none of the VQE circuits can be executed within the coherence time of the Brooklyn system.

While the number of threshold values and the precision factor are highly relevant parameters for the quality of the solution, they also contribute to the number of quadratic terms in the QUBO

matrix, making both quantum annealing and gate-based quantum computing less feasible as shown by the simulation results. As such, the manner in which these parameters should be configured for specific join ordering problems in order to achieve a sufficient solution quality remains an open yet important question and is a topic for future research.

In addition, future research should focus on expanding the join ordering MILP model by more sophisticated cost functions and other extensions discussed in [16]. Obviously, the inclusion of such extensions will lead to an additional overhead in the required qubits. However, as the quantum systems which can be accessed become more mature, it might be possible to solve even more extensive problem formulations on future devices.

Moreover, this work only considered the presented two-step transformation from join ordering to QUBO problems. It might, however, be possible to find a direct conversion without first transforming the problem into an MILP problem. Such an approach has the potential to be more efficient in terms of required qubits and thus be more feasible for current and future quantum machines. As such, future research should also focus on finding alternatives to the proposed two-step approach.

Finally, this work only considered the minorminer tool for determining embeddings on D-Wave quantum annealing machines. Finding different alternative algorithms for determining embeddings has been a topic of recent research. For instance, a method based on integer programming was presented in [32]. Using potentially more effective algorithms for determining embeddings can make it possible to solve join ordering problems of larger dimensions in comparison to the results which were gained by using the minorminer tool.

8 Conclusion and Outlook

This work addressed two research questions related to the use of quantum computing on problems in the field of query optimization. First, it analyzed the applicability of two hybrid quantum-classical algorithms for the MQO problem. More specifically, it compared simulation results for a current IBM-Q system with previous results achieved on a quantum annealer. It was found that significantly larger MQO problems can be solved on D-Wave quantum annealing systems in comparison to current IBM-Q devices. Regarding the two quantum-classical algorithms, the application of QAOA was found to be more practical than the VQE approach.

The second query optimization problem this work focused on was the join ordering problem. No pre-existing, direct method of formulating the problem in a way suitable for quantum systems was found after an extensive literature search. As such, the first goal was to identify a suitable reformulation method. As a result, a two-step reformulation approach was found and presented in this work. First, the problem can be formulated as an MILP problem based on an existing approach. After further adjustments, the problem can be transformed into a BILP problem, for which known transformations into QUBO form exist. The latter is suitable for current quantum systems.

Next, simulations have been conducted to evaluate the applicability of this two-step approach for an IBM-Q device and a D-Wave system. Again, the results showed that the latter can solve significantly larger problems in comparison to the former. However, even the D-Wave system does not come close to the possible problem dimensions classical MILP solvers can handle. Again, when considering the applicability of the quantum-classical algorithms, QAOA is superior to VQE as the latter produces circuit sizes which are too large for reliable computation whereas the former can be used to solve very small join ordering problems.

The limitations of the current era of NISQ quantum systems have once again been highlighted by the results of this work. Not only are the problem dimensions which can be tackled on current quantum systems very limited by the number of available qubits, additional limitations such as decoherence errors, which become more likely as the problems become more complex, need to be accounted for as well. However, as recent years have shown, quantum technologies are rapidly evolving and are steadily reaching a more mature state. In addition, the quality of the execution of quantum algorithms on future quantum systems will hopefully further increase with the prospect of quantum error correction.

As such, it might not be too long until systems become available which are suitable for solving problems of more practical dimensions. These problems include MQO as well as join ordering problems, which were discussed in this thesis, but may also include other database optimization problems so far not considered for quantum computing. Therefore, future research should focus on both identifying further problems suitable for quantum computing as well as determining proper reformulation approaches.

Bibliography

[1] L. K. Grover, "A fast quantum mechanical algorithm for database search", in *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing - STOC '96*, New York, New York, USA: ACM, 1996, pp. 212–219.

[2] P. W. Shor, "Algorithms for quantum computation: Discrete logarithms and factoring", in *Proceedings 35th Annual Symposium on Foundations of Computer Science*, Santa Fe, NM, USA: IEEE Comput. Soc. Press, 1994, pp. 124–134.

[3] IBM Quantum, *Cloud access to quantum computers provided by IBM*, 2021. [Online]. Available: `https://quantum-computing.ibm.com` (visited on 08/30/2021).

[4] J. Preskill, "Quantum computing in the NISQ era and beyond", *Quantum*, vol. 2, p. 79, 2018.

[5] J. R. McClean, J. Romero, R. Babbush, and A. Aspuru-Guzik, "The theory of variational hybrid quantum-classical algorithms", *New Journal of Physics*, vol. 18, no. 2, p. 023 023, 2016.

[6] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a photonic quantum processor", *Nature Communications*, vol. 5, no. 1, p. 4213, 2014.

[7] E. Farhi, J. Goldstone, and S. Gutmann, "A quantum approximate optimization algorithm", 2014. [Online]. Available: `arXiv:1411.4028`.

[8] C. McGeoch and P. Farré, "The D-Wave Advantage system: An overview", D-Wave Systems Inc., Tech. Rep. 14-1049A-A, 2020.

[9] I. Trummer and C. Koch, "Multiple query optimization on the D-Wave 2X adiabatic quantum computer", *Proceedings of the VLDB Endowment*, vol. 9, no. 9, pp. 648–659, 2016.

[10] G. Moerkotte, "Building query compilers", 2020, unpublished. [Online]. Available: `https://pi3.informatik.uni-mannheim.de/~moer/querycompiler.pdf`.

[11] M. Schönberger, "Applicability of quantum computing on database query optimization", in *SIGMOD'22: International Conference on Management of Data*, Philadelphia, NY, USA: ACM, in press.

[12] T. K. Sellis, "Multiple-query optimization", *ACM Transactions on Database Systems*, vol. 13, no. 1, pp. 23–52, 1988.

[13] T. Kathuria and S. Sudarshan, "Efficient and provable multi-query optimization", in *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, New York, NY, USA: ACM, 2017, pp. 53–67.

[14] M. A. Bayir, I. H. Toroslu, and A. Cosar, "Genetic algorithm for the multiple-query optimization problem", *IEEE Transactions on Systems, Man and Cybernetics, Part C (Applications and Reviews)*, vol. 37, no. 1, pp. 147–153, 2006.

[15] M. Steinbrunn, G. Moerkotte, and A. Kemper, "Heuristic and randomized optimization for the join ordering problem", *The VLDB Journal*, vol. 6, no. 3, pp. 191–208, 1997.

[16] I. Trummer and C. Koch, "Solving the join ordering problem via mixed integer linear programming", in *Proceedings of the 2017 ACM International Conference on Management of Data*, New York, NY, USA: ACM, 2017, pp. 1025–1040.

[17] M. A. Nielsen, I. Chuang, and L. K. Grover, "Quantum computation and quantum information", *American Journal of Physics*, vol. 70, no. 5, pp. 558–559, 2002.

[18] E. Rieffel and W. Polak, *Quantum computing: A gentle introduction*, ser. Scientific and engineering computation. Cambridge, MA: MIT Press, 2011.

[19] V. Choi, "Minor-embedding in adiabatic quantum computation: I. the parameter setting problem", *Quantum Information Processing*, vol. 7, no. 5, pp. 193–209, 2008.

[20] A. Lucas, "Ising formulations of many NP problems", *Frontiers in Physics*, vol. 2, p. 5, 2014.

[21] Z. Bian, F. Chudak, W. Macready, and G. Rose, "The Ising model: Teaching an old problem new tricks", D-Wave Systems Inc., Tech. Rep., 2010.

[22] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser, "Quantum computation by adiabatic evolution", 2000. [Online]. Available: `arXiv:quant-ph/0001106`.

[23] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev, "Adiabatic quantum computation is equivalent to standard quantum computation", *SIAM Review*, vol. 50, no. 4, pp. 755–787, 2008.

[24] S. Boixo, T. F. Rønnow, S. V. Isakov, Z. Wang, D. Wecker, D. A. Lidar, J. M. Martinis, and M. Troyer, "Evidence for quantum annealing with more than one hundred qubits", *Nature Physics*, vol. 10, no. 3, pp. 218–224, 2014.

[25] P. Murali, D. C. Mckay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers", in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA: ACM, 2020, pp. 1001–1016.

[26] S. S. Tannu and M. K. Qureshi, "Not all qubits are created equal: A case for variability-aware policies for NISQ-era quantum computers", in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, New York, NY, USA: ACM, 2019, pp. 987–999.

[27] L. Burgholzer, R. Raymond, and R. Wille, "Verifying results of the IBM Qiskit quantum circuit compilation flow", in *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, Denver, CO, USA: IEEE, 2020, pp. 356–365.

[28] P. I. Bunyk, E. M. Hoskinson, M. W. Johnson, E. Tolkacheva, F. Altomare, A. J. Berkley, R. Harris, J. P. Hilton, T. Lanting, A. J. Przybysz, and J. Whittaker, "Architectural considerations in the design of a superconducting quantum annealing processor", *IEEE Transactions on Applied Superconductivity*, vol. 24, no. 4, pp. 1–10, 2014.

[29] D-Wave Systems Inc., *Documentation for the architecture of D-Wave quantum annealers*, 2021. [Online]. Available: `https://docs.dwavesys.com/docs/latest/c_gs_4.html` (visited on 09/23/2021).

[30] S. Zbinden, A. Bärtschi, H. Djidjev, and S. Eidenbenz, "Embedding algorithms for quantum annealers with Chimera and Pegasus connection topologies", in *High Performance Computing*, Cham: Springer International Publishing, 2020, pp. 187–206.

[31] D-Wave Systems Inc., *Minorminer library for embedding Ising problems onto quantum annealers*, 2021. [Online]. Available: `https://docs.ocean.dwavesys.com/en/stable/docs_minorminer/source/intro.html` (visited on 10/04/2021).

[32] D. E. Bernal, K. E. C. Booth, R. Dridi, H. Alghassi, S. Tayur, and D. Venturelli, "Integer programming techniques for minor-embedding in quantum annealers", in *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, Cham: Springer International Publishing, 2020, pp. 112–129.

[33] S. Cluet and G. Moerkotte, "On the complexity of generating optimal left-deep processing trees with cross products", in *Database Theory — ICDT '95*, Berlin, Heidelberg: Springer Berlin Heidelberg, 1995, pp. 54–67.

[34] IBM Quantum, *Qiskit: An open-source framework for quantum computing*, 2021. [Online]. Available: `https://qiskit.org/` (visited on 08/31/2021).

[35] Anaconda Inc., *Anaconda software distribution*, 2021. [Online]. Available: `https://docs.anaconda.com/` (visited on 08/31/2021).

[36] IBM, *IBM decision optimization CPLEX modeling for Python*, 2021. [Online]. Available: `https://ibmdecisionoptimization.github.io/docplex-doc/` (visited on 10/04/2021).

[37] M. Schönberger, M. Franz, S. Scherzinger, and W. Mauerer, "Peel | pile? Cross-framework portability of quantum software", in *19th IEEE International Conference on Software Architecture (ICSA)*, Honolulu, HI, USA: IEEE, under review.

[38] M. Conforti, G. Cornuéjols, and G. Zambelli, *Integer programming*, ser. Graduate Texts in Mathematics. Cham: Springer International Publishing, 2014, vol. 271.

[39] C. S. Calude and M. J. Dinneen, "Solving the broadcast time problem using a D-Wave quantum computer", in *Advances in Unconventional Computing: Volume 1: Theory*, Cham: Springer International Publishing, 2017, pp. 439–453.

[40] C.-Y. Chang, E. Jones, and P. Graf, "On quantum computing for mixed-integer programming", 2020. [Online]. Available: `arXiv:2010.07852`.

[41] B. O'Gorman, R. Babbush, A. Perdomo-Ortiz, A. Aspuru-Guzik, and V. Smelyanskiy, "Bayesian network structure learning using quantum annealing", *The European Physical Journal Special Topics*, vol. 224, no. 1, pp. 163–188, 2015.

[42] D-Wave Systems Inc., *Documentation for the Ocean SDK for solving problems on D-Wave quantum computers*, 2021. [Online]. Available: `https://docs.ocean.dwavesys.com/en/stable/` (visited on 10/15/2021).

[43] Gurobi Optimization, LLC, *Gurobi optimizer reference manual*, 2021. [Online]. Available: `https://www.gurobi.com` (visited on 08/25/2021).

[44] D-Wave Systems Inc., *Pyqubo package for creating QUBO models from mathematical expressions*, 2021. [Online]. Available: `https://docs.ocean.dwavesys.com/en/stable/docs_pyqubo.html` (visited on 08/25/2021).