Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Quantum Computing for Smart Energy Optimizations

Fabian Weik

Course of Study:	Informatik
Examiner:	Prof. Dr. Marco Aiello
Supervisor:	Daniel Vietz, M. Sc.
Commenced:	October 12, 2020
Completed:	April 12, 2020

Kurzfassung

Diese Arbeit erkundet die Nützlichkeit des Quanten Computings für das Planen der Stromgenerierung thermischer Kraftwerke. Da immer mehr Smart Meter, also smarte Stromzähler, in Haushalten verbaut werden, kann die Energieindustrie den Energieverbrauch genauer und mit einer höheren Zeitauflösung vorhersagen. Dadurch werden die mathematischen Modelle größer. Der Anstieg von Erneuerbaren Energien trägt auch zur weiteren Vergrößerung der Modelle bei. Bisherige Optimierungen dieser Planungen involviert sogenannte Mixed-integer Non-linear Problems. Für die Optimierung dieser Modelle sind nur Algorithmen mit einer sehr hohen Zeitkomplexität bekannt. Die hier vorgestellte Methode formuliert das Problem zu einem Diskreten Quadratischen Modell. Dieses wird dann von einem hybriden klassisch-quanten Algorithmus optimiert. Diese Methode reduziert die Zeitkomplexität aber reduziert gleichzeitig die Genauigkeit des Ergebnisses. Diese Arbeit vergleicht die Methode mit einem klassischen open-source Algorithmus der zur Optimierung von Mixed-integer Non-linear Problems genutzt wird. Beide Algorithmen werden auf realitätsnahen Daten getestet.

Abstract

This paper explores the use of Quantum Computing for the scheduling of thermal power plants. With the rise of Smart Meters, the energy industry can forecast the energy demand more precisely and with better time resolution. This increased time resolution increases the size of the mathematical models used to schedule the power plants. Also, the rise of renewable energy sources further increases the size of the models. Current techniques for the precise optimization of these models are computationally expensive and have high time complexity. They involve Mixed-integer Non-linear Problems. Quantum Computing is a powerful tool for optimizations. The proposed method reformulates the Mixed-integer Non-linear Problem into a Discrete Quadratic Model. A hybrid classical-quantum algorithm then optimizes this model. The method decreases the time complexity but also loses some precision in finding the true optimum. This paper compares its performance and precision with a classical open-source algorithm used for optimizing Mixed-integer Non-linear Problems. The comparison involves running both algorithms on real-world data.

Contents

1 Introduction **15**

2 Fundamentals **17**
2.1 Unit Commitment Problem 17
2.2 Mixed-Integer Non-linear Problems 18
2.3 Quantum Computing 18
2.4 Related Work 24

3 Approach **25**
3.1 Unit Commitment Problem as Mixed-Integer Nonlinear Problem 25
3.2 Unit Commitment Problem as Discrete Quadratic Model 26
3.3 Unit Commitment Problem as Quadratic Unconstrained Binary Model 31

4 Implementation **37**
4.1 Implementation on Classical Computers 37
4.2 Implementation for Annealing-based Quantum Computers 39
4.3 Implementation for Gate-based Quantum Computers 45
4.4 Converting Solutions of Quadratic Models 46

5 Evaluation **49**
5.1 Data used for Validation 49
5.2 Performance of Classical Computers 52
5.3 Performance of Annealing-based Quantum Computers 53
5.4 Performance of Gate-based Quantum Computers 57
5.5 Comparison of Performances 57

6 Conclusion **61**

Bibliography **63**

List of Figures

1.1 Sketch of Unit Commitment Problem 16

2.1 Deutsch Algorithm for $f : x \mapsto x$ 21
2.2 Number of Qubits in D-wave QPUs over the Years [D-W18; D-W20a] 23
2.3 Characteristics of IBM Quantum Processors 23

5.1 Power Demand Data 50
5.2 Time Complexity of Classical Optimization with 4 Power Plants 53
5.3 Time Complexity of Annealing Optimization with 4 Power Plants 54
5.4 Minimum Computing Time of Hybrid DQM Solver 55
5.5 Minimum Computing Time of Hybrid QUBO Solver 56
5.6 Performance Comparison of Classical and Hybrid Annealing Algorithms 58
5.7 Relative Error of Hybrid Annealing Algorithm compared to Classical Algorithm 59

List of Tables

5.1 Characteristics of Power Plants 50
5.2 Possible Power Output Levels of the Power Plants 51
5.3 Results of Classical Optimization with 4 Power Plants 53
5.4 Results of Annealing Optimization with 4 Power Plants 54
5.5 Interpolation Points for Minimum Computing Time of Hybrid DQM Solver . . . 55
5.6 Interpolation Points for Minimum Computing Time of Hybrid DQM Solver . . . 56

Code Listings

4.1 Implementation of the Objective Function for MINLPs 38
4.2 Implementation of the Load Constraint for MINLPs 39
4.3 Implementation of the Power Constraint for MINLPs 39
4.4 Implementation of the Quadratic Startup and Shutdown Biases for DQMs 40
4.5 Implementation of the Quadratic Demand Biases for DQMs 41
4.6 Implementation of the Linear Biases for DQMs 41
4.7 Implementation of the Quadratic Startup and Shutdown Biases for QUBOs 42
4.8 Implementation of the Quadratic Demand Biases for QUBOs 43
4.9 Implementation of the Quadratic Discretization Biases for QUBOs 43
4.10 Implementation of the Linear Biases for QUBOs 44
4.11 Implementation of the Linear Startup and Shutdown Biases for QUBOs 44
4.12 Choosing Power Levels from the QUBO 46
4.13 Adjusting the UCP Solution 47

Nomenclature

$i \in \mathbb{I}$	Indices of all units $\{0, 1, \ldots, I\}$
$t \in \mathbb{T}$	Indices of all time steps $\{0, 1, \ldots, T\}$
$u_{i,t}$	Commitment of unit i at time t
$u_{i,-1}$	Initial commitment of unit i
$p_{i,t}$	Power output of unit i at time t
$f_{i,t}$	Cost function of unit i at time t
A_i	Constant cost function coefficient of unit i
B_i	Linear cost function coefficient of unit i
C_i	Quadratic cost function coefficient of unit i
A_i^U	Start-up cost of unit i
A_i^D	Shut-down cost of unit i
$s_{i,t}$	Actual start-up or shut-down cost of unit i at time t
$P_{min,i}$	Minimum power output of unit i
$P_{max,i}$	Maximum power ourput of unit i
l_t	Power demand (“load”) at time t

Discrete Quadratic Model

$\mathbb{M}$	Indices of all variables of the model
m	Mapping function of $\mathbb{I} \times \mathbb{T} \rightarrow \mathbb{M}$
P_i	Vector of possible power output levels of unit i
δ_i	Difference between the non-zero power levels of P_i
n_i	Defines number of elements in P_i: $\lvert P_i \rvert = 2^{n_i}$
E	Energy Function of DQM
γ_c	Factor of ojective function
γ_s	Factor of startup and shutdown constraints
γ_d	Factor of demand constraint
$p_{i,t}$	Vector that signals the power output of unit i at time t
F_i	Vector of costs of corresponding power output level P_i
$S_i^{(0)}$	Vector of startup or shutdown costs of unit i at time 0
S_i	Matrix of startup and shutdown costs of unit i
v_{DQM}	Number of variables of the model
q_{DQM}	Number of quadratic biases of the model

Quadratic Unconstrained Binary Optimization

$k \in \mathbb{N}_0, k < \lvert P_i \rvert$	Indices of possible power levels of plant i
$\mathbb{M}'$	Indices of all variables of the model
m'	Mapping function of $\mathbb{I} \times \mathbb{T} \times \{k \in \mathbb{N}_0 \lvert k < \lvert P_i \rvert\} \rightarrow \mathbb{M}'$
$P_{i,k}$	kth possible power output level of unit i
E	Energy function of QUBO
γ_p	Factor of discretization constraints
$p_{i,t,k}$	Variable that signals the power output of unit i at time t at level k
$F_{i,k}$	Costs of corresponding power output level $P_{i,k}$
$S_{i,k}^{(0)}$	Startup or shutdown cost of unit i at time 0 at level k
v_{QUBO}	Number of variables of the model
q_{QUBO}	Number of quadratic biases of the model

1 Introduction

The scheduling of conventional thermal power plants is crucial with the rise of renewable energies. The Unit Commitment Problem (UCP) tackles this problem [BMM+11]. Also, it can be used with wind energy and thermal power plants to minimize the cost of incorrect weather predictions. These approaches are also called risk-based because they model the risk of the weather predictions being wrong [AMJ17; Che08].

The number of deployed smart meters in the European Union was 99 million in 2018. A study by the European Commission projected the number to reach 123 million by 2020 [VØ09]. With the increasing number of smart meters, the amount of information about energy consumption is growing. The data is available with a time resolution that was not yet possible. Due to this large amount of data, predicting future power consumption is possible on a house-to-house basis. That allows for a much better overall prediction of the power consumption of towns or cities [AP16; BDB13]. It presents the possibility to schedule thermal power plants much more precisely and with a higher time resolution.

In the future, communities with many private renewable energy sources might rely on distributed energy generation. Distributed energy generation means that the energy produced by a consumer through their renewable energy sources gets added to the grid if the consumer is currently not using this energy [AP16]. Additionally, electric vehicles can be used as energy sources when plugged into a charger [ZHD+16]. The addition of these energy sources gives additional degrees of freedom for energy procurement.

Figure 1.1 is a sketch of the problem that the work considers. It shows a town with offices and houses, as well as charging stations for electric vehicles. There are 4 power plants around the town that are all connected to it.

Optimizing the UCP takes a lot of time for large inputs. Large inputs here means one of two things or both: (i) More units, i.e., power plants or (ii) A longer timeframe, over which to schedule the units. UCPs are optimization problems and thus, most of the time formulated as Mixed-integer Non-linear Problems [Bal95]. Precisely optimizing these problems is NP-complete [Bie96; LS05]. The energy industry needs faster algorithms for large inputs to deal with the increasing amount of information. These algorithms also have to be precise.

Quantum computing promises optimization algorithms with improved time complexity [AHY20; DH96; GWG19; PH00; SUN+19]. There are two types of quantum processors. One of which is designed for optimizations and the other which is a more general quantum processor.

Today's quantum computers are described as Noisy Intermediate-Scale Quantum (NISQ) technology. That is because the hardware is sensitive to noise and thus not able to run long computations. Also, there are not many qubits in the latest quantum computers (intermediate-scale) [LB20]. That is why one can't run most quantum algorithms for practical problems. Either there are not enough qubits to represent the variables, or the result is too noisy. One alternative that still can give a speed-up for

some problems are hybrid classical-quantum algorithms. Such algorithms solve the problem at hand with classical computers and use quantum computers to speed up specific parts that a NISQ quantum computer can compute [AHY20; SUN+19].

This work explores the application of purely quantum and hybrid classical-quantum algorithms to the UCP. It compares their performance in regards to computation time and accuracy of the result. The goal is to find an algorithm that can accurately solve large UCPs faster than the current classical algorithms.

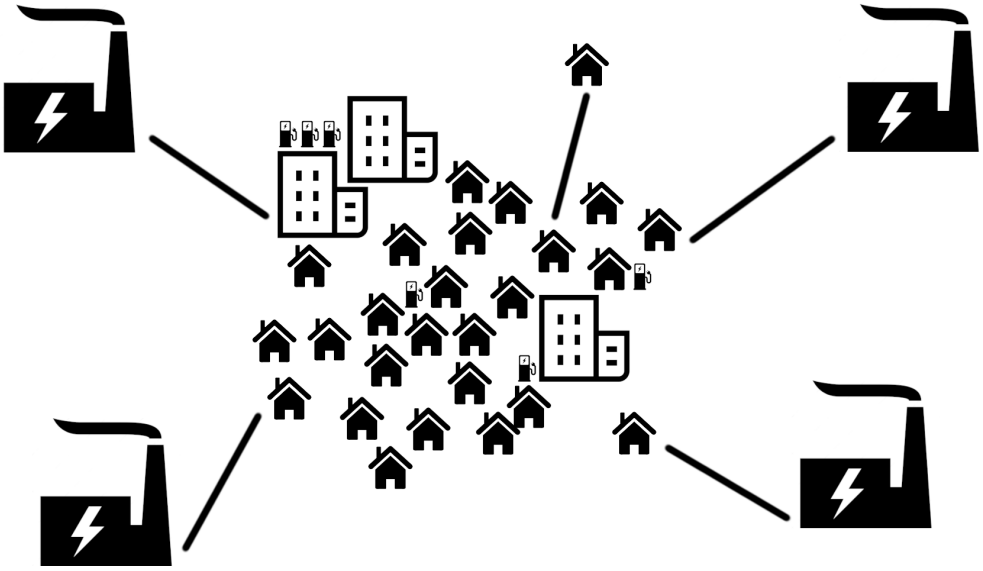


Figure 1.1: Sketch of Unit Commitment Problem

2 Fundamentals

2.1 Unit Commitment Problem

The Unit Commitment Problem (UCP) generally is concerned with the scheduling or "commitment" of thermal power plants over a given period. Thermal power plants can be coal, oil, gas, or nuclear power plants. The power plants should be scheduled such that the power demand or "network load" at each moment of the period is satisfied by the power output of the power plants. While satisfying the demand, the sum of the fuel costs ($f_{i,t}$) should be minimal. The fuel cost depends on the commitment of each power plant at each time ($u_{i,t}$), the power output of each power plant at each time ($p_{i,t}$), the power plant-specific cost function coefficients (A_i, B_i and C_i), as well as start-up and shut-down costs (A_i^U and A_i^D). Other than the demand, the problem also considers various other constraints of the power plants. These include the minimum and maximum power output of each plant ($P_{min,i}$ and $P_{max,i}$), the ramping constraints of each plant, and the minimum up and down-time [Bal95].

This work only considers a subset of all possible constraints. The demand, the minimum and maximum power output of the individual plant, and the startup and shutdown costs are considered. So the goal is to minimize the sum of the cost functions ($f_{i,t}$) that also considers the startup and shutdown-costs. Also, at each time instance, the combined power output satisfies the demand, and each power output does not violate the minimum and maximum power output of the power plant.

The UCP, this work considers, is non-linear because of the quadratic cost functions ($f_{i,t}$). It also has floating-point variables and binary variables, where binary variables are represented as integer variables with domain $\{0, 1\}$. The class of Mixed-Integer Non-Linear Problems (MINLP) is a class of optimization problems, with a non-linear objective function, integer and floating-point variables, and various constraints on the variables. For that reason, the UCPs often are formulated as MINLPs. These models are non-convex in the case of the UCP due to the binary nature of the commitment variables ($u_{i,t}$) [AMJ17; Bal95].

2.2 Mixed-Integer Non-linear Problems

Mixed-integer Non-linear Problems (MINLPs) are a class of optimization problems that fit a specific form. Formula (2.1) shows the from of MINLPs.

(2.1a)
$$\min_{x} \quad f(x)$$
(2.1b)
$$s.t. \quad g_j(x) \leq 0 \qquad \forall j \in M$$
(2.1c)
$$x_i^l \leq x_i \leq x_i^u \qquad \forall i \in N_0$$
(2.1d)
$$x_i \in \mathbb{Z} \qquad \forall i \in N_0^I \subseteq N_0$$

where $f(x)$ and all $g(x)$ may be non-linear. That is why the class of problems is called non-linear.

The first part (2.1a) is the objective function. The goal is to find an input $x \in \mathbb{R}^n$ that minimizes $f(x)$ while also respecting the constraints. The second part (2.1b) lists all the constraints that involve multiple elements of the input x. The input x has to comply with all constraints. The third part (2.1c) sets limits for the elements x_i of x. And the last part 2.1d states that some of the elements x_i are integers rather than rational numbers [BLL+09].

Optimizing MINLPs is NP-complete [Bie96]. That means that no known algorithm can optimize all possible MINLPs with polynomial time complexity.

2.3 Quantum Computing

Quantum computers use quantum bits (qubits) instead of bits that classical computers use. While a bit can be either in the state 0 or 1, a qubit can be in a so-called superposition of two states. States of qubits are generally written using the Dirac-Notation, also known as the Bra-Ket-Notation. The two states, the superposition is a combination of, are called the basis. The most basic basis is the basis $\{|0\rangle, |1\rangle\}$. The formula (2.2a) describes such a superposition. When one measures the state of the qubit, it collapses into one of the base states, which is the measurement result. The probability of either basis state being measured is the absolute square of the complex amplitude of the basic state in the superposition. Formula (2.2b) shows this for the state $|0\rangle$ [VP98].

(2.2a)
$$|\phi\rangle = \alpha \cdot |0\rangle + \beta \cdot |1\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} \qquad \text{with } \alpha, \beta \in \mathbb{C}, \alpha^2 + \beta^2 = 1$$
(2.2b)
$$p_0 = |\alpha|^2$$

Multiple qubits chained together form a quantum register. In such a register, multiple qubits might be entangled. That means that the measurement of one qubit depends on the measurement of the other qubit. The formula (2.3a) describes a quantum register with n qubits. If one can't separate the

formula into single qubits, the state is entangled. The probability of measuring a specific state of the quantum register is given by the absolute square of the amplitude of that state Formula (2.3b) shoes this for all states $|x\rangle$ [VP98].

(2.3a)

$$|\Phi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x \cdot |x\rangle = \begin{pmatrix} \alpha_{0\ldots00} \\ \alpha_{0\ldots01} \\ \alpha_{0\ldots10} \\ \ldots \\ \alpha_{1\ldots11} \end{pmatrix} \qquad \text{with } \forall x \in \{0,1\}^n : \alpha_x \in \mathbb{C}, \sum_{x\in\{0,1\}^n} \alpha_x^2 = 1$$

(2.3b)

$$p_x = \alpha_x^2$$

2.3.1 Annealing-based Quantum Computing

Annealing-based quantum computers are designed to tackle optimization problems well. The quantum processing unit (QPU) takes an Ising-model as input and embeds the biases on itself. For this, it uses couplers between its qubits. Then quantum tunneling is exploited to find a qubit-state where the system's energy is minimal. The result is the state of the Ising-model variables that minimize the model [BAS+13].

Ising models and QUBOs are equivalent, and the conversion is computationally inexpensive. The difference is the domain of the variables [BCMR10]. In Ising models, the domain is $\{-1, 1\}$, and in QUBOs, it is $\{0, 1\}$. This work considers QUBOs rather than Ising models. The formula (2.4) shows the structure of a QUBO. a are called linear biases because they depend on one variable, and b are called quadratic biases because they depend on two variables. c is called the constant bias because it depends on no variables and is constant.

(2.4) $$E(v) = \sum_i a_i \cdot v_i + \sum_{i<j} b_{i,j} \cdot v_i \cdot v_j + c$$

Hybrid Quantum-annealing

The number of qubits on a QPU is limited, and so are the couplings between them that the QPU uses to embed quadratic biases. When a problem has too many variables or quadratic biases, it's impossible to find an embedding for the QPU [BMF20].

A hybrid sampler is an algorithm that runs on a classical computer and a quantum computer. The classical computer receives the problem and attempts to solve it. It uses the quantum computer for subproblems that the quantum computer can solve faster [ZHD+16].

D-Wave's Advantage system has more than $5,000$ qubits with 15-way connectivity [D-W20a]. It can solve problems with up to $5,000$ binary variables if the quadratic biases are ≤ 15 for every variable. If there are some variables with more quadratic biases, they can be distributed onto different qubits. At some point, this is not possible anymore, and a hybrid algorithm must be used. D-Wave's hybrid sampler can handle fully connected QUBOs of up to $20,000$ variables. It can handle non-fully connected QUBOs of up to $1,000,000$ variables, but the number of biases has to be smaller than $200,000,000$ [BMF20].

Discrete Optimization

D-Wave also has a hybrid sampler that can handle problems with discrete variables instead of binary as in QUBOs. These problems have to be a Discrete Quadratic Model (DQM). Formula (2.5) shows the form of such DQMs. The sampler can handle a maximum of $5,000$ variables with up to $10,000$ values per variable. The maximum number of biases the sampler supports is $2,000,000,000$ [D-W20b].

$$E(v) = \sum_i a_i \cdot v_i + \sum_{i<j} b_{i,j} \cdot \left(v_i \otimes v_j\right) + c \tag{2.5}$$

Note that in the formula (2.5), a_i is a vector that holds the linear biases for the variable i. $a_i \cdot v_i$ is the scalar product of the linear biases with the value of variable i. $b_{i,j}$, on the other hand, is a matrix that holds the quadratic biases of the variable i and j. $b_{i,j} \cdot \left(v_i \otimes v_j\right)$ is the Frobenius Inner Product $\langle b_{i,j}, v_i \otimes v_j \rangle_F$ that is defined as the sum of all elements of both matrices multiplied element-wise.

2.3.2 Gate-based Quantum Computing

In the gate-based model of quantum computers, gates represent the manipulation of qubits. The qubits and quantum gates form a quantum circuit.

Single qubit gates manipulate only one qubit at a time. Double qubit gates manipulate two qubits and can entangle two qubits. The formula (2.6a) shows an example of a single qubit gate. This gate is called the Pauli-X gate, and it swaps the amplitudes of the two basis states $|0\rangle$ and $|1\rangle$. The formula (2.6c) shows an example of a double qubit gate. This gate is called the CNOT, and it applies the Pauli-X gate to the second qubit if the first qubit is in the base state $|1\rangle$. The CNOT gate entangles the involved qubits.

(2.6a)

$$X = |0\rangle\langle 1| + |1\rangle\langle 0| = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

(2.6b)

$$H = \frac{1}{\sqrt{2}} \left(\left(|0\rangle + |1\rangle\right) \langle 0| + \left(|0\rangle - |1\rangle\right) \langle 0| \right) = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

(2.6c)

$$CNOT = |00\rangle\langle 00| + |01\rangle\langle 01| + |10\rangle\langle 11| + |11\rangle\langle 10| = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

A quantum circuit consists of one or more quantum gates that are operating on one or more qubits. The quantum circuit 2.1 depicts the Deutsch algorithm for the function $f : x \mapsto x$. The dot with the line to the circled plus stands for the CNOT gate, and the H gates stand for Hadamard gates. A

Hadamard gate puts qubits that are in a basic state into a superposition where both outcomes of the measurement are equally likely. It is described by the formula (2.6b). The last gate stands for a measurement. In this case only the first qubit is measured [Deu85].

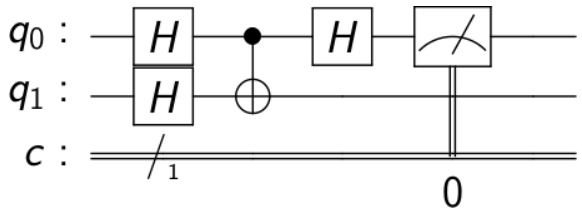


Figure 2.1: Deutsch Algorithm for $f : x \mapsto x$

A quantum computer that implements this model of computing would be more powerful than any Turing machine. It can simulate any finite physical system with polynomial time complexity, including systems with quantum effects. The complexity class is called Bounded-error Quantum Polynomial-time (BQP). There exists no known algorithm for Turing machines to accomplish this [Deu85; Sho98].

Grover Algorithm

The Grover Algorithm is a quantum algorithm for unstructured search. It can find an element that satisfies some condition C in an unstructured list with the time complexity of $O(\sqrt{n})$ [Gro96]. If multiple elements satisfy the condition, every execution of the algorithm yields only one of those elements [BBHT98].

The algorithm consists on 3 parts:

1. Initialize the input register with the equal superposition $|\Phi\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N} |x\rangle$
2. Repeat the following steps $O(\sqrt{N})$ times:
   a) Rotate all states that satisfy C by π.
   b) Apply the diffusion operator D. It is defined as

$$D_{i,j} = \begin{cases} \frac{2}{N} & i \neq j \\ \frac{2}{N} - 1 & else \end{cases} \tag{2.7}$$

3. Sample the input register.

Step 2 is called the Grover-iteration G. Its first part that rotates desired states is called an oracle. The Grover-iteration amplifies the amplitudes of desired states. This amplification increases the probability of measuring those states in step 3 [Gro96].

Optimization

Optimizing QUBOs on gate-based quantum computers is possible by iteratively searching for better values of the QUBO variables. The algorithm iteratively uses the Grover search to search for better values for the QUBO variables. It uses two quantum registers to represent the variables of the QUBO ($|x\rangle$), and the energy value of the QUBO ($|z\rangle$) [GWG19].

The oracle consists of multiple rotations of qubits of $|z\rangle$. The qubits of $|x\rangle$ control the rotations. The rotations add the biases of the QUBO to the value of the quantum register in the Fourier-domain. After that, the oracle subtracts the current minimum energy value y from the value of $|z\rangle$. The inverse Fourier-transformation converts the $|z\rangle$ register to a Two's Complement basis. The most significant qubit of $|z\rangle$ signals the sign of the value and controls the typical oracle used to rotate the desired states [GWG19].

This way, the measurement of $|x\rangle$ yields values for the QUBO variables that correspond to a lower energy value y' than the minimum found before y. In the next iteration, the oracle subtracts y' from $|z\rangle$ after adding the biases. By repeating this process, the algorithm identifies the values for the QUBO variables that minimize the energy [GWG19].

2.3.3 State of the art

Annealing-based Quantum Computers

D-Wave is the leading manufacturer of annealing-based quantum computers. Their newest model, "Advantage", has over $5,000$ qubits and over $35,000$ couplers. The connectors connect the qubits in a P_{16}-Graph, also called "Pegasus". This graph has the degree 16, which means that every qubit has a connection with 15 other qubits. D-Wave released this model in 2020 [D-W20a; ZBDE20].

The model before this, "2000Q", has over $2,000$ qubits and over $6,000$ couplers. The connectors connect the qubits in a C_{16}-Graph, also called "Chimera". This graph has the degree 6, which means that every qubit has a connection with 5 other qubits. D-wave released that model in 2017 [D-W20a; ZBDE20].

D-wave, therefore, increased their qubit number significantly in the last years. The qubit count increased by 150% in just 3 years. The coupler count increased by 580% in the same time frame, although this number is misleading. The more meaningful metric anout the coupler count is the qubit connectivity, the so-called "degree", which increased by 133%.

Figure 2.2 shows the increase of the number of qubits that are in D-Wave's QPUs. The number of qubits doubles all 2 years.

Gate-based Quantum Computers

IBM is the leading manufacturer of Gate-based Quantum Computers. Their newest model, "Hummingbird", has exactly 65 qubits. It debuted in 2020. Unfortunately, information about the performance of this processor is not available to the public [Gam].

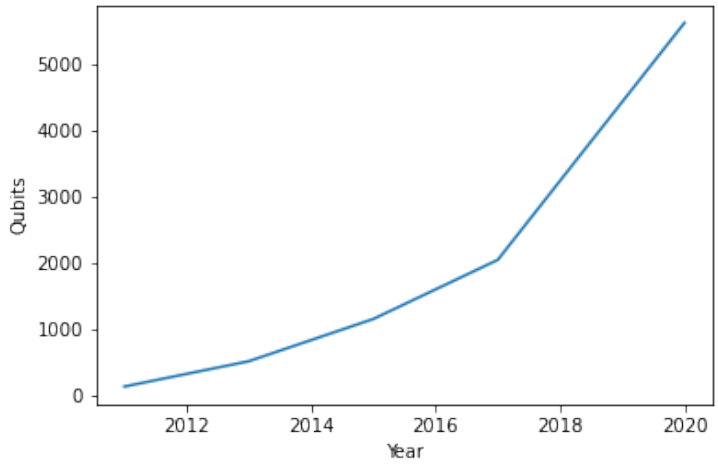
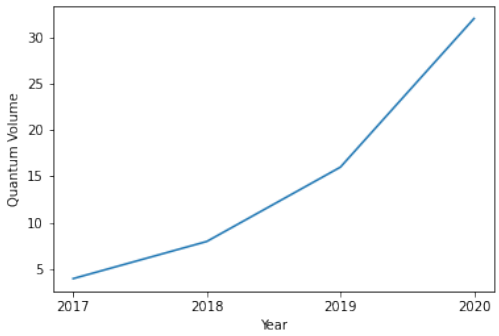


Figure 2.2: Number of Qubits in D-wave QPUs over the Years [D-W18; D-W20a]

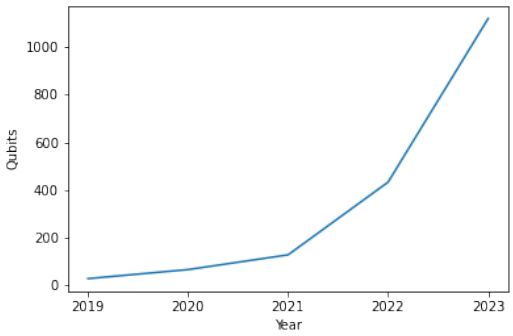
The model before that, "Falcon", has exactly 27 qubits. It debuted in 2019. IBM was able to achieve a quantum volume of 64 on this processor [Gam].

Quantum volume is a metric that characterizes the power of quantum processors. It depends on the number of qubits and the number of gates that a quantum circuit can have without producing significant errors [BBC+17]. IBM managed to achieve a 100% increase of the quantum volume metric every year since 2017 [GM]. This is illustrated in figure 2.3a.

Even though the quantum volume is an improved measurement of the power of a quantum processor, the number of qubits is still important. It determines how big the problems can be for the quantum processor to solve them. IBM promises at least a doubling of the qubit count every year in their 2020 roadmap. It projects its 2023 quantum processors to have 1, 121 qubits. Figure 2.3b ilustrates the projected qubit count [Gam].



(a) Quantum Volume achieved of IBM Quantum Processors [GM]



(b) Projected Qubit Count of IBM Quantum Processors [Gam]

Figure 2.3: Characteristics of IBM Quantum Processors

2.4 Related Work

Many methods for optimizing the UCP described in section 2.1 exist in the literature. Abujarad et al. listed and compared those methods. They listed the use of Mixed-integer Linear Programming as accurate but exponentially expensive. All other methods give a sub-optimal solution [AMJ17].

When dealing with cost functions of thermal power plants, they seldom are linear. Most commonly, they are quadratic. Baldick generalized the formulation of the UCP as Mixed-integer Non-linear Problems, where he assumed a quadratic cost function.

Ajagekar and You explore the utility of quantum computing for optimization problems in the energy domain. They also considered the UCP. They formulated a UCP as a QUBO and used D-Wave quantum hardware — the D-Wave 2000Q, to be precise — to optimize that model of the UCP. Compared to the optimization on a classical computer using the Gurobi solver, the quantum sampler did perform a lot worse than the classical algorithm [AY19].

At the end of 2020, D-Wave released a hybrid solver that can solve Discrete Quadratic Models [D-W20b]. The DQM model has a large advantage over the QUBO model when modeling problems with discrete or continuous variables. The QUBO can only have binary variables, while the DQM can have discrete variables. Modeling discrete variables takes many binary variables and many more quadratic biases to ensure only one binary variable corresponding to one discrete variable is 1. This phenomenon also appears in the QUBO formulation of the UCP by Ajagekar and You.

This work attempts to solve the UCP on annealing and gate-based quantum hardware using purely quantum and hybrid classical-quantum algorithms. It considers the DQM formulation in addition to the QUBO formulation already tested by Ajagekar and You. It also compares the performance of the proposed algorithms versus a classical algorithm.

3 Approach

3.1 Unit Commitment Problem as Mixed-Integer Nonlinear Problem

The UCP is formulated as an MINLP for classical computers to solve, as mentioned in section 2.1. The equations (3.1) represent the formulated MINLP.

(3.1a)

$$\min_{u,p} \quad \sum_{i\in\mathbb{I},t\in\mathbb{T}} f_{i,t} = \sum_{i\in\mathbb{I},t\in\mathbb{T}} u_{i,t}(A_i + B_i p'_{i,t} + C_i (p'_{i,t})^2) + s_{i,t}$$

(3.1b)

$$\text{s.t.} \quad l_t = \sum_{i\in\mathbb{I}} u_{i,t} p'_{i,t} \qquad \forall t \in \mathbb{T}$$

(3.1c)

$$P_{min,i} \leq p'_{i,t} \leq P_{max,i} \qquad \forall i \in \mathbb{I}, t \in \mathbb{T}$$

(3.1d)

$$s_{i,t} = \begin{cases} A_i^U & \text{if } u_{i,t} > u_{i,t-1} \\ A_i^D & \text{if } u_{i,t} < u_{u,t-1} \\ 0 & \text{else} \end{cases} \qquad \forall i \in \mathbb{I}, t \in \mathbb{T}$$

Objective Function

The formula (3.1a) represents the global cost function. This function has to be minimized and is called the "objective function". It is the sum of all cost functions $f_{i,t}$ of the individual power plants at every time step.

The individual cost funtions $f_{i,t}$ take $u_{i,t}$ and $p'_{i,t}$ as free variables. A_i, B_i and C_i are constant for every power plant. $f_{i,t}$ also depend on $s_{i,t}$ or equation (3.1d), which evaluates to A_i^U, A_i^D or 0 depending on the free commitment variables $u_{i,t}$ and $u_{i,t-1}$. A_i^U and A_i^D are also constant for every power plant, as well as $u_{i,-1}$.

A_i, A_i^U and A_i^D are the linear coefficients of the cost function because they are not a factor of $p_{i,t}$. A_i is the constant cost when the power plant is active and is added to $f_{i,t}$ if $u_{i,t} = 1$. A_i^U is the cost of starting the power plant and is added to $f_{i,t}$ if $u_{i,t} > u_{i,t-1}$. A_i^D is the cost of shutting down the power plant and is added to $f_{i,t}$ if $u_{i,t} < u_{i,t-1}$. If $t = 0$ this would lead to problems, because $u_{i,t-1} = u_{i,-1}$ would be undefined. That's why the initial state of the power plant $u_{i,-1}$ is also

constant. B_i is the linear coefficient of the cost function because it is a factor of $p'_{i,t}$. It is multiplied by $p'_{i,t}$ and added to $f_{i,t}$ if $u_{i,t} = 1$. C_i is the quadratic coefficient of the cost function because it is a factor of $p^2_{i,t}$. It is multiplied by $p^2_{i,t}$ and added to $f_{i,t}$ if $u_{i,t} = 1$.

So the objective function takes the free variables $(u_{i,t})_{i\in\mathbb{I},t\in\mathbb{T}}$, $(p'_{i,t})_{i\in\mathbb{I},t\in\mathbb{T}}$ and the constants $(A_i)_{i\in\mathbb{I}}$, $(A_i^U)_{i\in\mathbb{I}}$, $(A_i^D)_{i\in\mathbb{I}}$, $(B_i)_{i\in\mathbb{I}}$, $(C_i)_{i\in\mathbb{I}}$, and $(u_{i,-1})_{i\in\mathbb{I}}$ as input. The choice of the free variables has to minimize the objective function.

Constraints

As the optimizer computes the optimal input to minimize the objective function, it has to make sure the constraints are satisfied These are given as equations or inequations. In this case there is one equation (3.1b) and one inequation (3.1c).

The equation (3.1b) makes sure that the combined power output equals the power demand at each time instance t. It is modeled by setting each constant $(l_t)_{t\in\mathbb{T}}$ equal to the sum of all power outputs at that time t. The summands on the right-hand side are the multiplication of $u_{i,t}$ and $p'_{i,t}$ and not simply $p'_{i,t}$ because, in this model, the power output can be greater than 0 when the unit is not active. The fact that the there exists a binary state $u_{i,t}$ in addition to $p'_{i,u}$ is a modeling trick to enable the inequation (3.1c).

The inequation (3.1c) makes sure, that the power output $p'_{i,t}$ of each power plant i at each time intance t is within the limits of the power plant. These limits are $P_{min,i}$ for the minimum and $P_{max,i}$ for the maximum power output. As mentioned earlier, the power output $p'_{i,t}$ never reaches 0, if $P_{min,i}$ is greater than 0. So after finding the minimum inputs u and p' for the MINLP problem, the solution to the UCP is the actual power $p_{i,t} = u_{i,t}p'_{i,t}$.

3.2 Unit Commitment Problem as Discrete Quadratic Model

The annealing-based hybrid classical-quantum optimizer requires a reformulation of the UCP as a DQM. This approach converts the MINLP formulation from section 3.1 to a DQM. As mentioned in section 2.3.1, DQM is an extension of QUBO by allowing for discrete variables rather than binary variables.

3.2.1 Map Variables

The model, as described in section 2.3.1, takes in a vector rather than a 2-dimensional matrix. That's why the indices of the variables $p_{i,t}$ are mapped, such that the result is just as many variables p_l.

$$m: \quad \mathbb{I}\times\mathbb{T}\to\mathbb{M}: \quad m(i,t) = i\cdot|\mathbb{T}| + t \tag{3.2}$$

In the model below, $p_{i,t}$ is used, but this is to show that

$$\begin{aligned} &\forall i\in\mathbb{I}, t, t'\in\mathbb{T}, t<t' : m(i,t) < m(i,t') \\ \wedge\quad &\forall i, j\in\mathbb{I}, t\in\mathbb{T}, i<j : m(i,t) < m(j,t) \end{aligned} \tag{3.3}$$

When implementing the DQM, the discrete variables $p_{i,t}$ are added in the order specified by $m(i,t)$. This order ensures that the applying of quadratic biases does not violate the form of the DQM. When adding a quadratic bias, the first argument is always a variable with a lower index $m(i,t)$.

3.2.2 Discretize Power Levels

The approach discretizes the power levels since the model respectively the variables are discrete. Every variable $p_{i,t}$ can have one of 2^{n_i} values. n_i is chosen so that the difference δ_i between the non-zero power levels represented by the values of $p_{i,t}$ is $5 < \delta_i \leq 10$. The power levels of the discrete values are represented as P_i. Their values are:

$$P_i = \begin{pmatrix} 0 \\ P_{min,i} \\ P_{min,i} + \delta_i \\ P_{min,i} + 2 \cdot \delta_i \\ \vdots \\ P_{max,i} \end{pmatrix} \tag{3.4}$$

This automatically enforces the constraint (3.1c) that ensures that the power outputs stay inside the limits of every power plant. This constraint thus is not found in the energy function (3.5).

Initial Fromula

The formula we start with (3.5) includes the objective function and every constraint of the MINLP formulation (3.1). Except for the constraint (3.1c) that makes sure the output of every power plant is within its limits. This is already enforced by the discretization of the power levels.

The formula is not in the standard form (2.5). The last summand must be multiplied out and the quadratic, linear and constant biases have to be grouped toghether.

(3.5a)

$$E(p) = \gamma_c \sum_{i,t} F_i p_{i,t}$$

(3.5b)

$$+ \gamma_s \sum_i \left(S_i^{(0)} p_{i,0} + \sum_{t>0} S_i \left(p_{i,t-1} \otimes p_{i,t} \right) \right)$$

(3.5c)

$$+ \gamma_d \sum_t \left(\left(\sum_i P_i p_{i,t} \right) - l_t \right)^2$$

3.2.3 Rewrite Objective Function

The first summand of the energy function (3.5a) represents the objective function (3.1a) without the startup and shutdown costs. The objective function takes the sum of the costs of operating the power plants at every time instance. The function can be modeled by a vector that holds the cost of operating a power plant at the power level of the corresponding index the power level is at in the vector P_i. It is achieved by applying the cost function used in the objective function (3.1a) element wise to P_i:

$$F_i = \begin{pmatrix} 0 \\ A_i + B_i \cdot P_{min_i} + C_i \cdot \left(P_{min,i}\right)^2 \\ A_i + B_i \cdot \left(P_{min_i} + \delta_i\right) + C_i \cdot \left(P_{min,i} + \delta_i\right)^2 \\ A_i + B_i \cdot \left(P_{min_i} + 2 \cdot \delta_i\right) + C_i \cdot \left(P_{min,i} + 2 \cdot \delta_i\right)^2 \\ \vdots \\ A_i + B_i \cdot \left(P_{max_i}\right) + C_i \cdot \left(P_{max,i}\right)^2 \end{pmatrix} \tag{3.6}$$

The multiplication of the vectors F_i and $p_{i,t}$ in the term (3.5a) represents the scalar product $\langle F_i, p_{i,t} \rangle$, which is the addition of all elements multiplied element-wise. Since only one element of the vector $p_{i,t}$ is 1, all other elements are 0, the resulting value is one element of F_i. Specifically, the result of the scalar product is the element of F_i that corresponds to the 1-element in $p_{i,t}$. F_i is called the linear bias of $p_{i,t}$.

3.2.4 Rewrite Startup and Shutdown Costs

The second summand of the energy function (3.5b) represents the startup and shutdown costs (3.1d). When a power plant i starts, it costs the plant a specified amount of fuel A_i^U. When it shuts down, it costs the plant the specified amount A_i^D.

The initial startup and shutdown costs can be modeled by a vector that holds either the startup costs or the shutdown costs. So for the variables $p_{i,0}$ the term (3.5b) produces an addition to the linear bias. $S_i^{(0)}$ depends on the initial state $u_{i,-1}$ of the power plant i and represents either the startup cost or the shutdown cost of the plant. The startup costs are only non-zero in the elements representing a power output greater than zero, so every element other than the zeroth. While the shutdown cost is only relevant in the zeroth element because that element corresponds to a shutdown plant.

$$S_i^{(0)} = \begin{cases} \left(0, A_i^U, \ldots, A_i^U\right)^T & , u_{i,-1} = 0 \\ \left(A_i^D, 0, \ldots, 0\right)^T & , u_{i,-1} = 1 \end{cases} \tag{3.7}$$

The startup and shutdown costs to later time instances can be modeled by a matrix that holds both the startup and shutdown costs of one plant. For all other variables, so $p_{i,t}$ for $t > 0$, the term (3.5b) does not produce linear biases, but rather quadratic ones. These are biases that depend on two variables. Here they depend on $p_{i,t-1}$ and $p_{i,t}$. There is one matrix for every power plant i. This

matrix holds the startup costs A_i^U at the elements in the zeroth row and column greater than zero. While the shutdown costs A_i^D are at the elements in the zeroth column and row greater than zero.

$$S_i = \begin{pmatrix} 0 & A_U & \dots & A_U \\ A_D & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_D & 0 & \dots & 0 \end{pmatrix} \tag{3.8}$$

The multiplication of the matrices S_i and $p_{i,t-1} \otimes p_{i,t}$ represents the Frobenius Inner Product $\langle S_i, p_{i,t-1} \otimes p_{i,t} \rangle_F$, which is similar to the scalar product of vectors. It is the sum of the element-wise products of both matrices. One element of each vector $p_{i,t-1}$ and $p_{i,t}$ has the value 1, so only one element of their tensor product has the value 1. Since all other elements of the tensor product are zero, the result of the inner product is one element of S_i. Specifically, the result of the inner product is the element of S_i that corresponds to the 1-element in the tensor product. This element represents the change of power output of the power plant i at the time step from $t-1$ to t. That is why the matrix S_i holds the startup costs at the elements that correspond to zero output in $t-1$ and non-zero output in t and vice versa. S_i is called quadratic bias.

3.2.5 Rewrite Demand

The third summand of the energy function (3.5c) represents the demand constraints (3.1b). It is formulated such that its value is the sum of the squared differences of the required power and the power output of all power plants at time t. So it gets bigger as the plants put out more or less than the required power. This formula has to be transformed so linear and quadratic biases for the variables can be formulated.

$$\begin{aligned} & \gamma_d \sum_t \left(\sum_i \left(P_i p_{i,t} \right) - l_t \right)^2 \\ = \; & \gamma_d \sum_t \left(l_t^2 + \sum_i \left(P_i p_{i,t} \right)^2 - l_t \sum_i \left(P_i p_{i,t} \right) + \sum_{i<j} \left(\left(P_i p_{i,t} \right) \left(P_j p_{j,t} \right) \right) \right) \end{aligned} \tag{3.9}$$

After multiplying out the product, respectively, the square, everything that is not a factor of any variables forms the constant bias. In this case it is

$$\gamma_d \sum_t l_t^2 \tag{3.10}$$

The second and third summand of the multiplied out term (3.9) are factors of one variable. Thus they form linear biases. The biases for the second summand of (3.9) can be modeled by squaring P_i element-wise.

(3.11)
$$P_i^{\odot 2} = \begin{pmatrix} 0 \\ P_{min,i}^2 \\ \left(P_{min,i} + \delta_i\right)^2 \\ \left(P_{min,i} + 2 \cdot \delta_i\right)^2 \\ \vdots \\ P_{max,i}^2 \end{pmatrix}$$

The biases of the third summand of (3.9) can be modeled by multiplying l_t and P_i. Toghether thay are

(3.12)
$$\gamma_d \sum_{i,t} \left(P_i^{\odot 2} - l_t P_i\right) p_{i,t}$$

The last summand of the multiplied out term (3.9) is a factor of two variables. So it results in quadratic biases. These quadratic biases can be modeled by taking the tensor product of P_i and P_j. The quadratic biases are then multiplied with the tensor product of $p_{i,t}$ and $p_{j,t}$ by taking the inner product as described in section 3.2.4. This behaves exactly as the last summand of (3.9) and gives a nice quadratic bias.

(3.13)
$$P_i \otimes P_j = \begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & P_{min,i}P_{min,j} & P_{min,i}\left(P_{min_j} + \delta_j\right) & \dots & P_{min,i}\left(P_{max,j}\right) \\ 0 & \left(P_{min,i} + \delta_i\right)P_{min,j} & \left(P_{min,i} + \delta_i\right)\left(P_{min_j} + \delta_j\right) & \dots & \left(P_{min,i} + \delta_i\right)\left(P_{max,j}\right) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & P_{max,i}P_{min,j} & P_{max,i}\left(P_{min_j} + \delta_j\right) & \dots & P_{max,i}P_{max,j} \end{pmatrix}$$

3.2.6 Final Formula

All reformulated summands added back toghether form the final formula. Then the biases are ordered by their degree.

The quadratic biases are grouped in the first summand of the result (3.14b). They do not interfere with each other since the demand biases are situated along the i-axes, and the startup biases are situated along the t-axes.

The linear biases are grouped in the second summand of the result (3.14c). They overlap for every variable, so they are added together. The linear biases $S_i^{(0)}$ are only relevant to the variables of the first time instance $t = 0$, so they are added in a separate sum.

The constant biases are grouped in the last summand of the result (3.14d). This is simply a constant summand. This could be removed without changing the nature of the solution that minimizes the energy function.

(3.14a)

$$E(p) = \gamma_c \sum_{i,t} F_i p_{i,t} + \gamma_s \sum_i \left(S_i^{(0)} p_{i,0} + \sum_{t>0} S_i \left(p_{i,t-1} \otimes p_{i,t} \right) \right)$$
$$+ \gamma_d \sum_t \left(l_t^2 + \sum_i \left(P_i^{\odot 2} - l_t P_i \right) i p_{i,t} + \sum_{i<j} \left(P_i \otimes P_j \right) \left(p_{i,t} \otimes p_{j,t} \right) \right)$$

(3.14b)

$$= \gamma_s \sum_i \sum_{t>0} S_i \left(p_{i,t-1} \otimes p_{i,t} \right) + \gamma_d \sum_t \sum_{i<j} \left(P_i \otimes P_j \right) \left(p_{i,t} \otimes p_{j,t} \right) \qquad \text{(quadratic)}$$

(3.14c)

$$+ \sum_{i,t} \left(\gamma_c F_i + \gamma_d \left(P_i^{\odot 2} - l_t P_i \right) \right) p_{i,t} + \sum_i S_i^{(0)} p_{i,0} \qquad \text{(linear)}$$

(3.14d)

$$+ \gamma_d \sum_t l_t^2 \qquad \text{(constant)}$$

3.2.7 Model Size

The size of DQMs consists of 3 parts: (i) Number of variables, (ii) Number of linear biases, and (iii) Number of quadratic biases. The number of variables v_{DQM} is identical with the number of linear biases. Formula (3.15a) gives the number of variables. Formula (3.15b) gives the number of quadratic biases q_{DQM}. Note that every quadratic bias is a matrix of numbers and every linear bias is a vector of numbers.

(3.15a)

$$v_{\mathrm{DQM}} = |\mathbb{T}| \cdot |\mathbb{I}|$$

(3.15b)

$$q_{\mathrm{DQM}} = (|\mathbb{T}| - 1) \cdot |\mathbb{I}| + |\mathbb{T}| \cdot \frac{|\mathbb{I}| \cdot (|\mathbb{I}| - 1)}{2}$$

3.3 Unit Commitment Problem as Quadratic Unconstrained Binary Model

The formulation of the Quadratic Unconstrained Binary Model (QUBO) is based on the DQM formulated in section 3.2.2. Instead of a single discrete variable for every power plant i and time instance t, the QUBO has an array of binary variables for every possible power level in P_i for every power plant i and time t.

3.3.1 Mapping Variables

The variable indices are mapped to a single index to make sure the QUBO form is correct. In addition to the mapping in 3.2.1, this mapping also considers the power levels of each power plant. The mapping is extended by a third index k corresponding to the power level.

(3.16)

$$m' : \quad \mathbb{I} \times \mathbb{T} \times \{k \in \mathbb{N}_0 | k < |P_i|\} \rightarrow \mathbb{M}' : \quad m'(i,t,k) = \left(|\mathbb{T}| \sum_{j=0}^{i-1} |P_j| \right) + t \cdot |P_i| + k$$

In the model below, $p_{i,t,k}$ is used, but this is to show that

(3.17)

$$\begin{aligned} & \forall i \in \mathbb{I}, t \in \mathbb{T}, k, l \in \mathbb{N}_0, k < l < |P_i| : m'(i,t,k) < m'(i,t,l) \\ \wedge \quad & \forall i \in \mathbb{I}, t, t' \in \mathbb{T}, k \in \mathbb{N}_0, k < |P_i|, t < t' : m'(i,t,k) < m'(i,t',k) \\ \wedge \quad & \forall i, j \in \mathbb{I}, t \in \mathbb{T}, k, l \in \mathbb{N}_0, k < |P_i|, l < |P_j|, i < j : m'(i,t,k) < m'(j,t,l) \end{aligned}$$

When implementing the QUBO, the discrete variables $p_{i,t,k}$ are added in the order specified by $m'(i,t,k)$. This order ensures that the applying of quadratic biases does not violate the form of the QUBO. When adding a quadratic bias, the first argument is always a variable with a lower index $m'(i,t,k)$.

3.3.2 Discretizising Power Levels

This approach discretizes the power levels just like in section 3.2.2. But this time, the values are not an array but single values. They are defined as

(3.18)

$$P_{i,k} = \begin{cases} 0 & , k = 0 \\ P_{min,i} + (k-1) \cdot \delta_i & , else \end{cases}$$

$P_{i,k}$ is defined for all $i \in \mathbb{I}, k \in \mathbb{N}_0, k < 2^{n_i}$. Where n_i is chosen as in section 3.2.2. Thus $P_{i,2^{n_i}-1} = P_{max,i}$.

This automatically enforces the constraint (3.1c) ensuring that the power outputs stay inside the limits of every power plant. That constraint thus is not found in the energy function (3.25).

3.3.3 Initial Formula

The starting point is the DQM (3.14). The approach transforms the formula in a way that it uses binary variables instead of discrete variables. Additionally, the QUBO has to ensure that only one of the binary variables corresponding to one power plant i at one time t is 1.

3.3.4 Rewrite Quadratic Biases

The transformation of the quadratic biases has two parts. The first part is the transformation of the startup and shutdown costs.

(3.19)
$$\gamma_s \sum_i \sum_{t>0} S_i \left(p_{i,t-1} \otimes p_{i,t}\right) = \gamma_s \sum_i \sum_{t,k>0} \left(A_i^D \cdot p_{i,t-1,k} \cdot p_{i,t,0} + A_i^U \cdot p_{i,t-1,0} \cdot p_{i,t,k}\right)$$

Penalties A_i^U and A_i^D apply if the power level of the plant i is 0 at time $t-1$ and non-zero at time t respectively non-zero at time $t-1$ and 0 at time t.

The second part is the transformation of the quadratic biases connected to the demand.

(3.20)
$$\gamma_d \sum_t \sum_{i<j} \left(P_i \otimes P_j\right) \left(p_{i,t} \otimes p_{j,t}\right) = \gamma_d \sum_{i,k,l} \sum_{i<j} \left(P_{i,k} \cdot P_{j,l} \cdot p_{i,t,k} \cdot p_{j,t,l}\right)$$

It is more straightforward and shows the meaning of the tensor products and the Frobenius inner product used in the DQM.

3.3.5 Rewrite Linear Biases

The transformation of the linear biases has two parts. The first part is the transformation of the objective and demand related biases.

(3.21)
$$\sum_{i,t} \left(\gamma_c \cdot F_i + \gamma_d \left(P_i^{\odot 2} - l_t \cdot P_i\right)\right) p_{i,t} = \sum_{i,t,k} \left(\gamma_c \cdot F_{i,k} + \gamma_d \left(P_{i,k}^2 - l_t \cdot P_{i,t}\right)\right)$$

Where $F_{i,k}$ is defined as follows:

(3.22)
$$F_{i,k} = \begin{cases} 0 & , k = 0 \\ A_i + B_i \cdot P_{i,k} + C_i \cdot P_{i,k}^2 & , else \end{cases}$$

The second part of the transformation is the startup and shutdown costs of the first time instance.

(3.23)
$$\sum_{i,k} S_{i,k}^{(0)} \cdot p_{i,0,k} \qquad \text{where } S_{i,k}^{(0)} = \begin{cases} A_i^U & , u_{i,-1} = 0 \wedge k > 0 \\ A_i^D & , u_{i,-1} = 1 \wedge k = 0 \\ 0 & , else \end{cases}$$

3.3.6 Discretize $p_{i,t,k}$

Since the variables in the QUBO are binary, each represents a single possible power level k of power plant i at time t. Each power plant i may only have one output level at each time instance t. So the QUBO should avoid that multiple binary variables for different power levels k and l are 1 for the same power plant i at the same time t. Penalizing multiple possible power output values for each plant at every time avoids this.

(3.24a)
$$\gamma_p \sum_{i,t} \left(\left(\sum_k p_{i,t,k} \right) - 1 \right)^2$$

(3.24b)
$$= \gamma_p \sum_{i,t} \left(1 + \sum_k \left(p_{i,t,k} \right)^2 - \sum_k \left(p_{i,t,k} \right) + \sum_{k<l} \left(p_{i,t,k} \cdot p_{i,t,l} \right) \right)$$

(3.24c)
$$= \gamma_p \sum_{i,t} \left(1 + \sum_{k<l} \left(p_{i,t,k} \cdot p_{i,t,l} \right) \right)$$

3.3.7 Final Formula

After adding all parts together again, we obtain the following QUBO.

(3.25a)
$$E(p) = \gamma_s \sum_i \sum_{t,k>0} \left(A_i^D \cdot p_{i,t-1,k} \cdot p_{i,t,0} + A_i^U \cdot p_{i,t-1,0} \cdot p_{i,t,k} \right) + \gamma_d \sum_{i,k,l} \sum_{i<j} \left(P_{i,k} \cdot P_{j,l} \cdot p_{i,t,k} \cdot p_{j,t,l} \right) + \gamma_p \sum_{i,t} \sum_{k<l} \left(p_{i,t,k} \cdot p_{i,t,l} \right) \quad \text{(quadratic)}$$

(3.25b)
$$+ \sum_{i,t,k} \left(\gamma_c \cdot F_{i,k} + \gamma_d \left(P_{i,k}^2 - l_t \cdot P_{i,k} \right) \right) p_{i,t,k} + \sum_{i,k} \left(S_{i,k}^{(0)} \cdot p_{i,0,k} \right) \quad \text{(linear)}$$

(3.25c)
$$+ \gamma_p + \gamma_d \sum_t l_t^2 \quad \text{(constant)}$$

3.3.8 Model Size

The size of QUBOs consists of the same 3 parts as DQMs (section 3.2.7): (i) Number of variables, (ii) Number of linear biases, and (iii) Number of quadratic biases. The number of variables v_{QUBO} is identical to the number of linear biases. Formula (3.26a) gives the number of variables. Formula

(3.26b) gives the number of quadratic biases. With this model, the linear and quadratic biases are both plain numbers.

(3.26a)

$$v_{\text{QUBO}} = \quad |\mathbb{T}| \cdot \sum_{i \in |\mathbb{I}|} 2^{n_i}$$

(3.26b)

$$q_{\text{QUBO}} = \quad (|\mathbb{T}| - 1) \cdot 2 \sum_{i \in |\mathbb{I}|} 2^{n_i} + |\mathbb{T}| \cdot \left(\sum_{i,j \in \mathbb{I}, i \neq j} 2^{n_i} \cdot 2^{n_j} + \sum_{i \in \mathbb{I}} \frac{2^{n_i} \cdot (2^{n_i} - 1)}{2} \right)$$

4 Implementation

4.1 Implementation on Classical Computers

For the implementation of the MINLP for classical computers, this work uses the Python library Pyomo to build the MINLPs from the UCPs. Pyomo is open-source and supports the formulation of various optimization problems, particularly mixed-integer nonlinear problems. It also supports performing the optimization of the problems with standard open-source or commercial solvers [HWW11].

The program is given a UCP object that holds information about the power demands and the available power plants. It then creates a "ConcreteModel" from this information. This model of the MINLP formulation of the UCP has three $|\mathbb{I}| \times |\mathbb{T}|$ matrices as variables. The first two are the variables for the commitment $u_{i,t}$ and the power output $p_{i,t}$ for every power plant at every time. These are the free variables of the model that will be adjusted so that the objective function is minimized. The last matrix is for $s_{i,t}$, the startup and shutdown costs. These are not free but depend on the data of the UCP object.

4.1.1 The Objective Function

The objective function (3.1a) is modeled using an `Objective` object provided by the Pyomo framework. It depends on the model variables for $u_{i,t}$, $p_{i,t}$ and $s_{i,t}$. Where $u_{i,t}$ and $p_{i,t}$ are free variables and $s_{i,t}$ are defined through the model. The coefficients A_i, B_i, and C_i are taken from the UCP object. Listing 4.1 shows the code that implements the objective function based on the free variables and the variables $s_{i,t}$ here called `startup_shutdown_cost`.

The startup and shutdown costs are defined through a method called "disjunctive programming". This method makes it possible to formulate "either-or" conditions and set values of variables based on the value of other variables [Bal83].

With this method the value of $s_{i,j}$ set to either A_i^U, A_i^D or 0 depending on the values of $u_{i,t}$ and $u_{i,t-1}$. This relation is defined mathematically by the constraint (3.1d). And it is modeled by giving the model three disjunct events for each plant i and time instance t:

$$\begin{aligned} & \left(u_{i,t} < u_{i,t-1} \wedge s_{i,t} = A_i^U\right) \\ \vee & \left(u_{i,t} > u_{i,t-1} \wedge s_{i,t} = A_i^D\right) \\ \vee & \left(u_{i,t} = u_{i,t-1} \wedge s_{i,t} = 0\right) \end{aligned} \tag{4.1}$$

```
def build_objective(self) -> None:
  '''
  builds the objective function of the MINLP
  '''
  def objective_function(model: ConcreteModel) -> Expression:
    plants: List[CombustionPlant] = self.ucp.plants

    return sum( # for every plant i
      sum( # and every time t
        model.u[(i, t)] * (
          plants[i].A +
          plants[i].B * model.p[i, t] +
          plants[i].C * model.p[i, t] ** 2
        ) + (
          model.startup_shutdown_cost[i, t]
        ) for t in model.T
      ) for i in model.I
    )

  self.model.o = Objective(rule=objective_function)
```

Code Listing 4.1: Implementation of the Objective Function for MINLPs

If $t = 0$, $u_{i,t-1}$ is defined as the initial commitment state of the power plant i. The first part of the events is the condition and the second part defines the value of $s_{i,t}$. Since the conditions are mutually exclusive, only one of the events will be true. That means that $s_{i,t}$ has the value specified on the right side of the true condition.

4.1.2 Constraints

The first constraint is the load constraint. It is specified by the formula (3.1b). It makes sure that the power demand is met by all the power plants i at every time t. The program creates a constraint list of length $|\mathbb{T}|$. Each element of the list makes sure that the sum of all power outputs at its corresponding time t is equal to the power demand at that time instance. Listing 4.2 shows the code that implements this constraint.

The second constraint is the power level constraint. It is specified by the formula (3.1c). It makes sure that the power output of every power plant i is within its limits at every time t. The program creates a constraint matrix of dimensions $|\mathbb{T}| \times |\mathbb{I}|$. Each element of the constraint matrix makes sure the power output of the corresponding power plant i at the corresponding time t is within the limits of the power plant's constraints. Listing 4.3 shows the code that implements this constraint.

```
def build_load_constraints(self) -> None:
  '''
  builds the constraints for the MINLP that make sure the power plants produce enough energy
  '''
  def load_constraint_rule(model: ConcreteModel, t: int) -> Expression:
    '''
    defines the rule for the constraints

    :model: MINLP
    :t: time index
    '''
    return self.ucp.loads[t] == sum(model.u[(i, t)] * model.p[(i, t)] for i in model.I)

  self.model.l_constr = Constraint(self.model.T, rule=load_constraint_rule)
```

Code Listing 4.2: Implementation of the Load Constraint for MINLPs

```
def build_power_constraints(self) -> None:
  '''
  builds the constraints for the MINLP that make sure every power output is inside the limits of the
                                                    power plants
  '''
  def power_constraint_rule(model: ConcreteModel, i: int, t: int) -> Expression:
    '''
    defines the rule for the constraints

    :model: MINLP
    :i: plant index
    :t: time index
    '''
    return inequality(self.ucp.plants[i].Pmin, model.p[(i, t)], self.ucp.plants[i].Pmax)

  self.model.p_constr = Constraint(self.model.I, self.model.T, rule=power_constraint_rule)
```

Code Listing 4.3: Implementation of the Power Constraint for MINLPs

4.2 Implementation for Annealing-based Quantum Computers

4.2.1 Implementation of DQM for Annealing Hardware

For the implementation of the DQM for annealing-based hardware, this work uses D-Wave's `dwave-ocean-sdk` [D-Wb]. To be more specific, it uses the class `DiscreteQuadraticModel` that is part of the `dimod` package [D-Wa]. The source code of the SDK is available online [D-Wc].

Variables

The first step is to instantiate a DQM with discrete variables. One discrete variable is added to the model for every $p_{i,t}$ of the DQM defined in section 3.2.2. The size of the discrete variable $p_{i,t}$ — the number of possible values — is $|P_i|$.

```
def set_quadratic_startup(self, y_s: float) -> None:
  '''
  sets the quadratic biases for the startup costs for the DQM
  '''
  for i in range(self.ucp.parameters.num_plants):
    # compute once for every plant i
    plant: CombustionPlant = self.ucp.plants[i]
    num_cases: int = len(self.P[i])
    quadratic_biases: np.ndarray = np.zeros((num_cases, num_cases))

    for k in range(1, num_cases):
      quadratic_biases[0, k] = plant.AU
      quadratic_biases[k, 0] = plant.AD

    quadratic_biases *= y_s

    for t in range(1, self.ucp.parameters.num_loads):
      # apply for one plant i at every time t > 0
      self.model.set_quadratic(self.p[i][t-1], self.p[i][t], quadratic_biases)
```

Code Listing 4.4: Implementation of the Quadratic Startup and Shutdown Biases for DQMs

Quadratic Biases

There are 2 types of quadratic biases — biases resulting from either (i) Startup and shutdown costs, or (ii) Demand constraints.

The startup and shutdown biases are different for every plant but do not change over time. The program computes them once for every plant. Then it applies them to variables that correspond to the same plant but correspond to adjacent time instances. So S_i is computed and then applied to variables $p_{i,t-1}$ and $p_{i,t}$ for all $t > 0$. Listing 4.4 shows the code implementing these quadratic biases.

The demand biases are different for every pair of plants involved but do not change over time. The program computes them once for every pair of plants. Then it applies them to the variables that correspond to the same time t but correspond to each one of the involved plants. So $P_i \otimes P_j$ is computed and then applied to the variables $p_{i,t}$ and $p_{j,t}$ for all $i < j$. Listing 4.5 shows the code implementing these quadratic biases.

The two categories of quadratic biases do not overlap. This is because the startup and shutdown biases are applied along the time axes, and the demand biases are applied along the plant axes, which is orthogonal to the time axes.

Linear Biases

The linear biases are different for every plant and every time instance. They depend on the fuel cost, the possible power levels, and the demand at that time instance. If the variable is of the form $p_{i,0}$ — meaning it corresponds to the first time instance — the linear bias of that variable is also dependent on the startup or shutdown costs and the initial state of the plant. The formula (3.14c) captures the computation of the linear biases.

```
def set_quadratic_demand(self, y_d: float) -> None:
  '''
  sets the quadratic biases for the demand for the DQM
  '''
  for j in range(1, self.ucp.parameters.num_plants):
    for i in range(j):
      # compute once for every pair of plants i, j (i < j)
      quadratic_biases: np.ndarray = np.tensordot(self.P[i], self.P[j], axes=0)
      quadratic_biases *= y_d

      for t in range(self.ucp.parameters.num_loads):
        # apply for one pair of plants i, j at every time t
        self.model.set_quadratic(self.p[i][t], self.p[j][t], quadratic_biases)
```

Code Listing 4.5: Implementation of the Quadratic Demand Biases for DQMs

```
  def set_linear(self, y_c: float, y_s: float, y_d: float) -> None:
    '''
    sets the linear biases for the DQM
    '''
    for i in range(len(self.p)):
      P_i: np.ndarray = self.P[i]
      plant: CombustionPlant = self.ucp.plants[i]
      F_i: np.ndarray = self.calculate_F_i(plant, i)

      for t in range(len(self.p[i])):
        linear_biases: np.ndarray = y_c * F_i + y_d * (
          P_i * P_i - self.ucp.loads[t] * P_i
        ) # implements formula for linear biases of the report

        if t == 0: # if t is 0, add initial startup or shutdown costs
          if plant.initially_on:
            linear_biases[0] += y_s * plant.AD

          else:
            for k in range(1, len(linear_biases)):
              linear_biases[k] += y_s * plant.AU

        self.model.set_linear(self.p[i][t], linear_biases)
```

Code Listing 4.6: Implementation of the Linear Biases for DQMs

The program iterates over all power plants i and time instances t. It calculates the linear bias as specified in the formula (3.14c) for all indices. Then it applies the calculated linear bias to the variable $p_{i,t}$. Listing 4.6 shows the code implementing these linear biases.

Constant Bias

The constant bias is defined by the fromula (3.14d). The class `DiscreteQuadraticModel` does not have an interface for adding a constant bias. Because a constant bias does not change the optimal input value, the implementation ignores it.

```
def add_quadratic_startup_shutdown(self, y_s: float) -> None:
  '''
  sets the quadratic biases for the startup costs for the QUBO
  '''
  for i in range(self.ucp.parameters.num_plants):
    # compute once for every plant i
    AU: float = self.ucp.plants[i].AU
    AD: float = self.ucp.plants[i].AD

    for t in range(1, self.ucp.parameters.num_loads):
      # apply for one plant i at every time t > 0
      for k in range(1, len(self.P[i])):
        self.add_quadratic_bias(i, t-1, 0, i, t, k, AU * y_s)
        self.add_quadratic_bias(i, t-1, k, i, t, 0, AD * y_s)
```

Code Listing 4.7: Implementation of the Quadratic Startup and Shutdown Biases for QUBOs

4.2.2 Implementation of QUBO for Annealing Hardware

For the implementation of the QUBO for the annealing-based hardware, this work uses the UQO-client [LMa; LMb]. First, it writes the biases into a Python `dict`. Then the class `Qubo` of the UQO-client builds the `dwave-ocean-sdk` `BinaryQuadraticModel`-instance that the UQO-client can then send to the UQO-server.

Variables

The variables in this implementation are implicit. The indices for the Python `dict` are pairs of integers where the integers are the IDs of the variables. Each ID is computed with the mapping function m' of the formula (3.16).

Quadratic Biases

In this case, there are 3 types of quadratic biases — biases resulting from either (i) Startup and Shutdown costs, or (ii) Demand constraints, or (iii) Discretization constraints.

The startup and shutdown costs are different for every power plant but do not change over time. The program iterates over all power plants i and times $t > 0$ and applies the startup and shutdown costs. The program iterates over every power output value index $0 \leq k < 2^{n_i}$ for the power plant i. It adds the quadratic bias A_U for the pair $(p_{i,t-1,0}, p_{i,t,k})$ and the quadratic bias for the pair $(p_{i,t-1,k}, p_{i,t,0})$. Listing 4.7 shows the code that implements these quadratic biases.

The demand constraints are different for every plant pair but do not change over time, like the startup and shutdown costs. The program iterates over all pairs of power plants (i, j) where $i < j$ and power output value indices $0 \leq k < 2^{n_i}$ and $0 \leq l < 2^{n_j}$. Then it iterates over all times t and applies the quadratic constraint $\gamma_d P_{i,k} * P_{j,l}$ to the pair $(p_{i,t,k}, p_{j,t,l})$ The quadratic constraint is only computed once for every pair of power plants (i, j) and pair of power output values (k, l). Listing 4.8 shows the code that implements these quadratic biases.

```
def add_quadratic_demand(self, y_d: float) -> None:
  '''
  sets the quadratic biases for the demand for the QUBO
  '''
  for j in range(self.ucp.parameters.num_plants):
    for i in range(j):
      for l in range(len(self.P[j])):
        for k in range(len(self.P[i])):
          # compute for every quadruplet i, j, k, l (i < j) (i, j plant indices, k, l power output level
                                                    indices)
          value: float = self.P[j][l] * self.P[i][k]
          for t in range(self.ucp.parameters.num_loads):
            # apply to every time t
            self.add_quadratic_bias(i, t, k, j, t, l, value * y_d)
```

Code Listing 4.8: Implementation of the Quadratic Demand Biases for QUBOs

```
  def add_quadratic_discretized(self, y_d: float) -> None:
    '''
    sets the quadratic biases for making sure only one power level is active per unit and time
    '''
    for i in range(self.ucp.parameters.num_plants):
      for t in range(self.ucp.parameters.num_loads):
        for l in range(len(self.P[i])):
          for k in range(l):
            # apply for every pair of power output levels k, l (k < l) for every plant i at every time t
            self.add_quadratic_bias(i, t, k, i, t, l, y_d)
```

Code Listing 4.9: Implementation of the Quadratic Discretization Biases for QUBOs

The discretization constraint is the same for every plant and time instance. The program iterates over all power plants i and times t. Then it iterates over all pairs of power output value indices (k, l) where $0 \leq k < 2^{n_i}, 0 \leq l < 2^{n_j}$ and $k < l$ and aplies the quadratic constraint γ_d to the pair $(p_{i,t,k}, p_{i,t,l})$. Listing 4.9 shows the code that implements these quadratic biases.

The quadratic biases do not overlap. The demand constraint biases and startup and shutdown biases are on orthogonal axes as mentioned in section 4.2.1. The biases constraining the power levels per power plant — only one power level k per power plant i and time instance t may be chosen — are between different power levels $0 \leq k < 2^{n_i}, 0 \leq l < 2^{n_i}$ with $k \neq l$ of the same power plant at the same time instance t. Therefore they don't interfere with the demand constraint biases since they are between different power levels $0 \leq k < 2^{n_i}, 0 \leq l < 2^{n_j}$ of different power plants i, j.

Linear Biases

The linear biases depend on the fuel cost and the demand constraints. That's why they change for every plant i, every time instance t, and every possible power level k. Just as with the implementation of the DQM in section 4.2.1, the linear biases of variables of the form $p_{i,0,k}$ depend on the startup and shutdown costs and the initial state of the power plant i. The formula (3.25b) captures the computation of the linear biases.

```
def add_linear(self, y_c: float, y_d: float) -> None:
  '''
  sets the linear biases for the QUBO
  '''
  for i in range(self.ucp.parameters.num_plants):
    plant: CombustionPlant = self.ucp.plants[i]
    for t in range(self.ucp.parameters.num_loads):
      for k in range(1, len(self.P[i])):
        value: float = 0

        # implements the formula for linear biases of the report
        value += y_c * (plant.A + plant.B * self.P[i][k] + plant.C * (self.P[i][k] ** 2))
        value += y_d * (self.P[i][k] ** 2 - self.ucp.loads[t] * self.P[i][k])

        self.add_linear_bias(i, t, k, value)
```

Code Listing 4.10: Implementation of the Linear Biases for QUBOs

```
def add_linear_startup_shutdown(self, y_s: float) -> None:
  '''
  sets the linear biases for the QUBO regarding startup and shutdown costs
  '''
  for i in range(self.ucp.parameters.num_plants):
    # compute once for every plant i
    is_initially_on: bool = self.ucp.plants[i].initially_on

    if is_initially_on:
      A_D: float = self.ucp.plants[i].AD
      self.add_linear_bias(i, 0, 0, y_s * A_D)

    else:
      A_U: float = self.ucp.plants[i].AU
      for k in range(1, len(self.P[i])):
        self.add_linear_bias(i, 0, k, y_s * A_U)
```

Code Listing 4.11: Implementation of the Linear Startup and Shutdown Biases for QUBOs

The program iterates over all power plants i, times t, and possible power levels $0 \leq k < 2^{n_i}$. It calculates the linear bias as specified in formula (3.25b) for all indices. Then it applies the linear bias to the variable $p_{i,t,k}$. Listing 4.10 shows the code that implements these linear biases.

The program computes the linear biases regarding the initial startup and shutdown costs independently. These biases have to be applied for different power levels k, depending on the initial state of the power plant. Including this in the algorithm 4.10 would lead to hard to read code. The program iterates over all pants i. Then it decides, based on the initial state of the plant, whether to apply the startup costs or the shutdown costs at time $t = 0$. For the startup costs, the program applies A_i^U to the variable $p_{i,0,0}$. For the shutdown costs, the program applies A_i^D to all variables $p_{i,0,k}$ with $k > 0$. Listing 4.11 shows the code that implements these linear biases.

As mentioned in section 4.2.2, the program first builds the QUBO as a Python `dict`. The keys are pairs of integers (i, j) with $i \neq j$ for quadratic biases. For linear biases, the keys are pairs with identical integers (i, i).

Constant Bias

The constant bias is defined by the formula (3.25c). Because the constant bias does not change the optimal input value, the implementation ignores it.

4.3 Implementation for Gate-based Quantum Computers

The implementation for the gate-based quantum computers uses the Qiskit framework. Qiskit is an open-source framework for quantum computing on gate-based quantum computers. IBM Research founded Qiskit [Qisa; Qisb]. After the program instantiates the QUBO using the `QuadraticProgram` class of the framework, it uses the class `GroverOptimizer` to optimize the QUBO. This class handles executing the Grover Optimization algorithm that section 2.3.2 describes. That includes building the quantum circuits of Grover algorithm instances for every iteration and the classical computation in between iterations.

This section omits the code snippets because they are the same as in section 4.2.2. The only difference is how the program adds the biases to the model because it uses a different framework. The biases have the same values and are applied to the same variables.

Variables

In this implementation, the variables are not implicit as with the UQO-client in section 4.2.2. The program instantiates all binary variables of the QUBO at the beginning of building the QUBO instance. It assigns a name to every variable. That name holds the indices of the corresponding power plant i, time t, and power output level k.

Quadratic Biases

The program adds the same quadratic biases as the implementation for annealing-based hardware that is described in section 4.2.2. It stores the biases in a Python `dict` with keys that are pairs of strings. The strings are the names of the variables.

Linear Biases

The program adds the same linear biases as the implementation for annealing-based hardware that is described in section 4.2.2. It stores the biases in a Python `dict` with keys that are strings. The strings are the names of the variables.

```
# store every power level k possible at time t for plant i
value_indices: List[int] = []
for k in range(len(self.P[i])):
  if result[self.m[i, t, k]] == 1:
    value_indices.append(k)

value: float = 0
num_indices: int = len(value_indices)
if num_indices > 0:
  # choose median power level
  value = self.P[i][value_indices[(int) (num_indices / 2)]]
  if num_indices > 1:
    debug_msg('Warning: {} possible power levels for plant {} detected'.format(num_indices, i))

p[i].append(value)
```

Code Listing 4.12: Choosing Power Levels from the QUBO

Canstant Bias

As mentioned before in section 4.2.2, the constant bias does not change the input that produces an optimum of the QUBO. But with this framework, the interface allows for a constant bias to be added to the QUBO. The implementation computes the bias once, as specified in the formula (3.25c, and adds it to the QUBO instance.

4.4 Converting Solutions of Quadratic Models

4.4.1 Choosing Power Levels

For QUBOs, it can happen that for a single power plant at a single time instance, multiple binary variables for different power levels are 1. The program then has to choose one of the possible power levels where the binary variable is 1. Listing 4.12 shows the code that is used to choose one power level. The program always chooses the median value.

4.4.2 Adjusting Power Levels

The program translates the optimal solution of the DQM or QUBO to a solution for the UCP. This solution, however, is not legal. The sum of the power outputs at every time instance does not satisfy the demand. The program, therefore, raises the power outputs of all plants at every time instance uniformly. While doing so, it also avoids violating the power limits of the individual plants. Listing 4.13 shows the code that adjusts the power levels.

```
for t in range(self.ucp.parameters.num_loads):
  # set adjust bool for every power plant that is active
  adjust: List[bool] = [self.u[i][t] for i in range(self.ucp.parameters.num_plants)]
  # compute the power that is missing or too much
  delta: float = self.ucp.loads[t] - sum([self.p[i][t] for i in range(self.ucp.parameters.num_plants)])

  while True:
    if delta == 0 or not functools.reduce(lambda a,b: a or b, adjust):
      # repeat the loop until either the demand is met (delta == 0)
      # or no plant can be adjusted anymore
      break

    adjustment: float = delta / sum([1 if b else 0 for b in adjust])
    delta = 0

    for i in range(self.ucp.parameters.num_plants):
      if adjust[i]:
        # add a portion of the delta to every power output
        self.p[i][t] += adjustment
        Pmax: float = self.ucp.plants[i].Pmax
        Pmin: float = self.ucp.plants[i].Pmin

        # if the power output violates the limits of the power plant
        # set it to the limit and add the difference to the delta again
        if self.p[i][t] > Pmax:
          delta += self.p[i][t] - Pmax
          self.p[i][t] = Pmax
          adjust[i] = False

        elif self.p[i][t] < Pmin:
          delta += self.p[i][t] - Pmin
          self.p[i][t] = Pmin
          adjust[i] = False
```

Code Listing 4.13: Adjusting the UCP Solution

5 Evaluation

5.1 Data used for Validation

5.1.1 Consumers

This work considers real-world demand data of energy consumers. It takes the data from two different sources. (i) Smart meters that Georgievski et al. installed in an office, and (ii) Adaptive Charging Network (ACN) electric vehicle charging stations at the Californian Institute of Technology [LLL19; Tec].

The data from the smart meters spans about 8 months. It is from the year 2016. It has a time resolution of 10 seconds. This work transforms the data to have a time resolution of 10 minutes. It does so by adding up the energy demand of 10 minute periods and then dividing by 0.166 hours since the data is given as energy in kWh. The result is the power given in kW.

The data from the ACN charging stations spans the first 9 months of the year 2020. It consists of various charging sessions. The program identifies the energy demand by assuming the charger transfers energy to the car at one fixed rate in one session. This behavior is not the point of adaptive charging, but it gives a real-world-like representation of electric vehicle charging behavior. The time resolution of the resulting data is 10 minutes.

The program computes the average day of both datasets. Then it combines both datasets. The data from the smart meters is weighted $1,000 : 1$ with respect to the data from the ACN charging stations. This composition is due to the low energy demand of one office. The final data represents an average day of one electric vehicle charging site and $1,000$ offices with a 10 minute time resolution. Figure 5.1 shows how the energy demand behaves over the day.

5.1.2 Producers

The data for the power plants is put together from different sources for different characteristics. (i) Maximum output power from a list by the Bundesnetzagentur, (ii) Minimum power output from Schröder et al., (iii) The startup and shutdown cost from Kumar et al., and (iv) Cost function coefficients from Alrashidi et al.

The work chooses 4 plants from the list of german power plants. The maximum output for every power plant comes from this list. All chosen plants are coal-powered power plants [Bun]. 2 of the plants are categorized as “sub-critical” and the other 2 as “super-critical”. The difference between “sub-critical” and “super-critical” coal-powered power plants is the pressure level at which they operate. “sub-critical” power plants operate at pressure levels below 221 bar and produce about

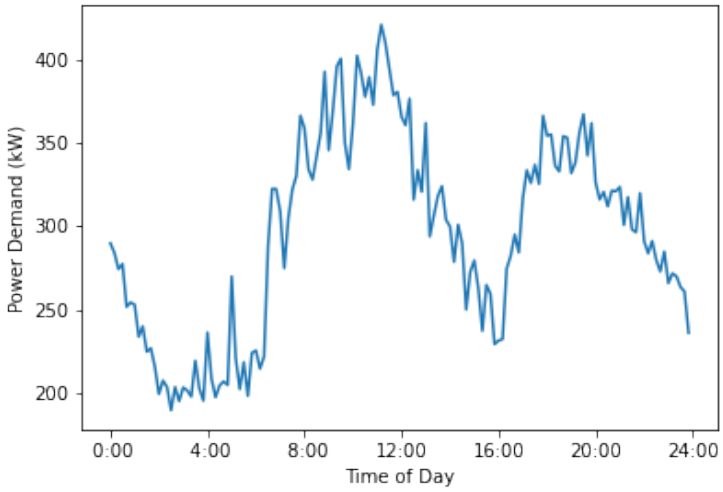


Figure 5.1: Power Demand Data

300 to 900 MW. "super-critical" power plants operate at pressure levels about 240 bar and produce between 500 to 1300 MW [KBL+12; SKMM13]. This choice is arbitrary but important for defining the other characteristics of the plants.

The minimum power output is computed based on the maximum power output and the type of power plant. For "sub-critical" power plants, the minimal power output is 35% of the maximum power output. For "super-critical" power plants, the minimal power output is 20% of the maximum power output [SKMM13].

The coal-powered power plants don't have shutdown costs. So only the startup costs are computed. For simplicity, this work only considers hot start startup costs. For "sub-critical" power plants, the startup cost is 7.5 MMBTU per MW maximum power output. For "super-critical" power plants, the startup cost is 10.1 MMBTU per MW maximum power output [KBL+12]. Where 1MMBTU = $1.6 \cdot 10^{-6}$GJ. This work considers all fuel costs in unit GJ.

The coefficients of the cost functions are taken from a paper that identified them at coal-powered power plants [AEA09]. Alrashidi et al. identified 2 polynomials both are assigned randomly to the 4 power plants. Table 5.1 lists the resulting values of all the described characteristics.

	Plant #1	Plant #2	Plant #3	Plant #4
Maximum Power (MW)	553	747	773	1900
Minimum Power (MW)	193.55	261.45	145.6	380
Startup Cost (GJ)	4.39635	5.93865	8.275738	20.3414
Constant Cost (GJ)	95.856	96.279	95.856	96.279
Linear Cost ($\frac{\text{GJ}}{MW}$)	7.374	7.592	7.374	7.592
Quadratic Cost ($\frac{\text{GJ}}{MW^2}$)	0.047	0.042	0.047	0.042

Table 5.1: Characteristics of Power Plants

5.1.3 Model Sizes

With the concrete values for the minimal and maximum power output of the power plants, the concrete size of the DQM and QUBO can be computed. For this the formulas (3.15a), (3.15b), (3.26a) and (3.26b) are used. The formulas give the size of the DQM and QUBO, respectively, based on the number of time instances, power plants, and the power output levels of the power plants. They consider 2 different metrics of the size: (i) The number of variables, which is the same as the number of linear biases ($v_{\text{DQM}}, v_{\text{QUBO}}$) and (ii) The number of quadratic biases ($q_{\text{DQM}}, q_{\text{QUBO}}$).

Size of DQM

For the DQM expanding the formulas (3.15a) and (3.15b), and inserting the number of power plants is straight forward and gives the formulas (5.1a) and (5.1b).

(5.1a)

$$v_{\text{QUBO}} = |\mathbb{T}| \cdot |\mathbb{I}| = 4 \cdot |\mathbb{T}|$$

(5.1b)

$$q_{\text{QUBO}} = (|\mathbb{T}| - 1) \cdot |\mathbb{I}| + |\mathbb{T}| \cdot \frac{|\mathbb{I}| \cdot (|\mathbb{I}| - 1)}{2} = 4 \cdot |\mathbb{T}| - 4 + 6 \cdot |\mathbb{T}| = 10 \cdot |\mathbb{T}| - 4$$

Size of QUBO

For the calculating the size of the QUBO, the number of discrete power output levels per power plant must be known. Section 3.3.2 describes the calculations nescessary to get this information. Table 5.2 shows the resulting numbers for the chosen power plants.

Power Plant i	n_i	2^{n_i}
0	6	64
1	6	64
2	7	128
3	8	256

Table 5.2: Possible Power Output Levels of the Power Plants

Inserting these values into the expanded formulas (3.26a) and (3.26b) gives the formulas (5.2a) and (5.2b).

(5.2a)

$$v_{\text{QUBO}} = |\mathbb{T}| \cdot \sum_{i \in \mathbb{I}} 2^{n_i} = |\mathbb{T}| \cdot (64 + 64 + 128 + 256) = 1024 \cdot |\mathbb{T}|$$

(5.2b)

$$\begin{aligned} q_{\text{QUBO}} &= (|\mathbb{T}| - 1) \cdot 2 \cdot \sum_{i \in \mathbb{I}} 2^{n_i} + |\mathbb{T}| \cdot \left(\sum_{i,j \in \mathbb{I}, i \neq j} 2^{n_i} \cdot 2^{n_j} + \sum_{i \in \mathbb{I}} \frac{2^{n_i} \cdot (2^{n_i} - 1)}{2} \right) \\ &= (|\mathbb{T}| - 1) \cdot (64 + 64 + 128 + 256) \\ &\quad + |\mathbb{T}| \left(2^{6+6} + 2^{6+7} \cdot 2 + 2^{6+8} \cdot 2 + 2^{7+8} + \frac{64 \cdot 63}{2} \cdot 2 + \frac{128 \cdot 127}{2} + \frac{256 \cdot 255}{2} \right) \\ &= 130,816 \cdot |\mathbb{T}| - 1,024 \end{aligned}$$

5.2 Performance of Classical Computers

For the optimization of the Pyomo model, the Pyomo library runs a standard solver. It formulates the problem in the mathematical modeling language AMPL and calls a solver to optimize it. The solver returns the solution, which Pyomo then reads [Lab].

This work uses the open-source solver Couenne for classical optimizations. COIN-OR built this solver. Couenne is short for “Convex Over and Under ENvelopes for Nonlinear Estimation”. It can find global minima of nonconvex MINLPs like the one that this work considers. It is a branch and bound algorithm and utilizes linearization, bound reduction, and branching methods [BLL+09; Foua; Foub].

This work performs the classical optimizations on the Excess Cluster of the HLRS, Stuttgart [HLR; Pro].

The optimization of the problem is of high time complexity. In the beginning, with small inputs, it takes little time. As listed in table 5.3, the optimization of 4 power plants with 2 units of time takes about 0.5 seconds. But as the input size grows, the time needed by the solver grows exponentially. When the input size is doubled — 4 plants over 4 units of time — the optimization takes about 3.5 seconds. After adding another 2 units of time, the optimization takes 15.5 seconds.

As the number of inputs grows large, the optimization grows even faster than exponentially. With a worst-case projection of the first 4 data points, the time needed for optimization should multiply by 7 with every additional 2 units of time. That means that the optimization of 4 power plants over 14 units of time should take about $0.5 \cdot 7^6 \approx 58,824$ seconds. But the data shows that it took about $311,144.9$ seconds.

Number of Loads	Objective function	Time (in seconds)	Time (in hours)
2	116037.074	0.517	0.000
4	224372.794	3.617	0.001
6	317725.099	15.582	0.004
8	405214.877	107.714	0.030
10	486041.406	1218.918	0.339
12	560470.679	17292.102	4.803
14	625146.685	311144.864	86.429

Table 5.3: Results of Classical Optimization with 4 Power Plants

The diagram 5.2 highlights the fast growing computation time needed for the optimization. Only the last two data points are distinguishable from the first data points because their computation time is very long compared to the first data points. It shows the time needed for optimization dependent on the input size — the number of time instances.

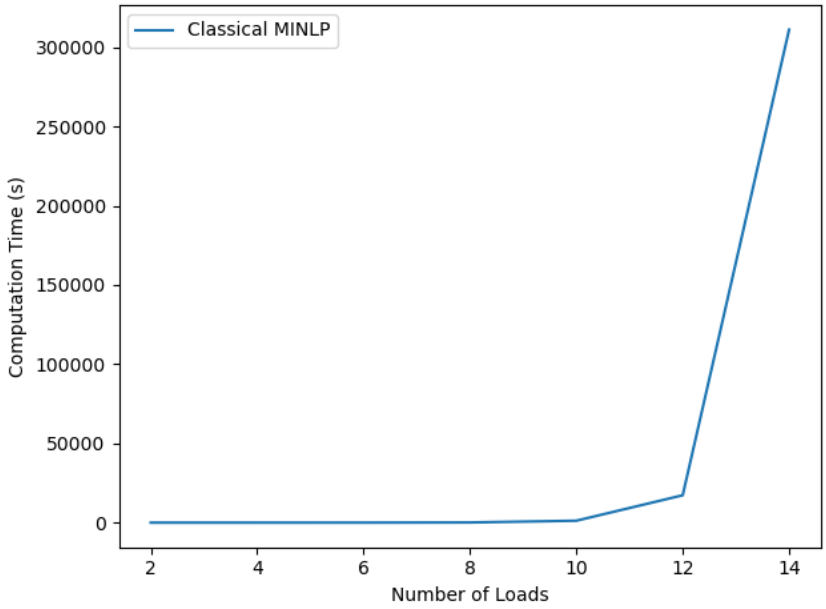
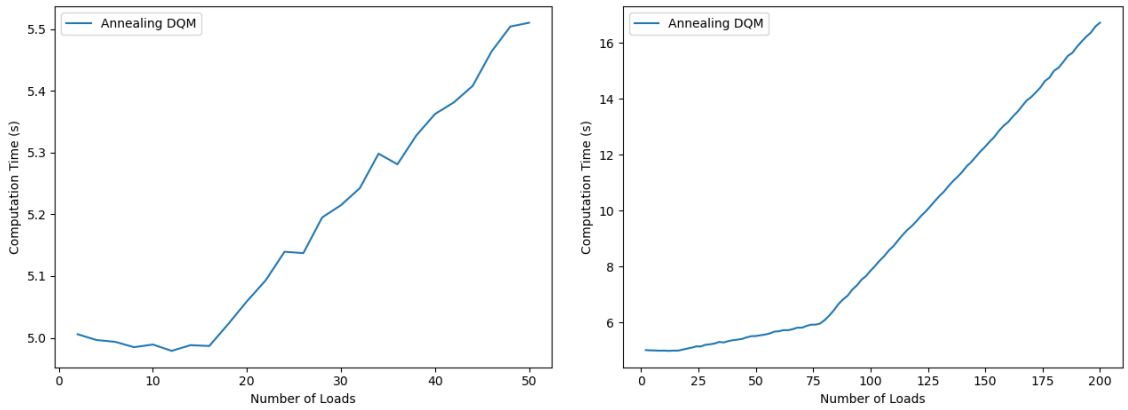


Figure 5.2: Time Complexity of Classical Optimization with 4 Power Plants

5.3 Performance of Annealing-based Quantum Computers

5.3.1 Hybrid DQM Solver

This work sends the formulated DQM to a hybrid sampler at D-Wave for the optimizations using annealing-based hardware. The hybrid sampler does not put the DQM on quantum hardware directly. It classically searches the solution space. The hybrid sampler speeds up this process by using quantum hardware to determine promising regions to explore next [D-W20b].



(a) With up to 50 Loads

(b) With up to 200 Loads

Figure 5.3: Time Complexity of Annealing Optimization with 4 Power Plants

When running the experiments, this work sets the parameters for tuning the DQM as 1. So $y_c = 1, y_s = 1, y_d = 1$. These parameters are part of the formula (3.14). One can change these to obtain better results. This work found that for small UCPs, this did not improve the quality of the solution.

The optimization of the DQM using the hybrid sampler for small problems seems to have linear time complexity. If the UCP grows by 10 time instances, the optimization takes about 0.1 seconds longer. Table 5.4 lists the results of the optimizations and the time the hybrid sampler needed to achieve that result. The optimization of these problems takes no longer than 5.6 seconds each.

Number of Loads	Objective function	Time (in seconds)
10	487007.496	4.989
20	817671.148	5.059
30	1155685.053	5.215
40	1554316.529	5.362
50	1559508.051	5.510

Table 5.4: Results of Annealing Optimization with 4 Power Plants

In figure 5.3a the time complexity is displayed clearer. The time needed for optimizing problems with less than 15 time instances also seems to be constant. After that, the linear time complexity is visible.

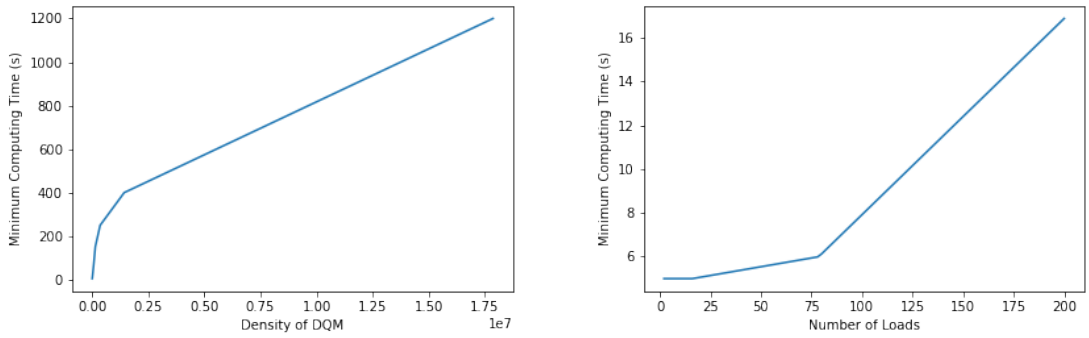
When the problem size increases, the time complexity seems to be worse than linear. This becomes clear in figure 5.3b. The time taken by the hybrid sampler to optimize the problem grows linearly but at a higher rate starting from 80 time instances. After this increase of slope, the slope stays the same until the problem size reaches 200 time instances. That is the maximum size of the UCP that this work considers on the hybrid DQM sampler.

The hybrid DQM sampler uses the minimum time specified by D-Wave for solving the problems. D-Wave defines the minimum computing time for the hybrid DQM sampler with the pairs of density and time listed in table 5.5. The sampler chooses the minimum computing time for a given

DQM by interpolating linearly between the nearest two points. Figure 5.4a shows the resulting function interpolating the points. Figure 5.4b shows the part of the function corresponding to the experimental data in figure 5.3b with the same unit — number of time instances instead of density. Figure 5.3b and figure 5.4b are very similar.

Density of DQM	Minimum Computing Time t
20000	5.0
100000	6.0
200000	13.0
500000	34.0
1000000	71.0
2000000	152.0
5000000	250.0
20000000	400.0
250000000	1200.0

Table 5.5: Interpolation Points for Minimum Computing Time of Hybrid DQM Solver



(a) Full **(b)** Corresponding to Extended Experiment Data

Figure 5.4: Minimum Computing Time of Hybrid DQM Solver

This approach does not produce optimal solutions for the UCP. The summed power output of the single power plants does not meet the power demand because the DQM underestimates the demand part of the UCP. After retrieving the optimal DQM solution and translating it to a UCP solution, the program adjusts the power outputs of the power plants. It does so as described in section 4.4.2. That leads to a non-optimal solution of the UCP.

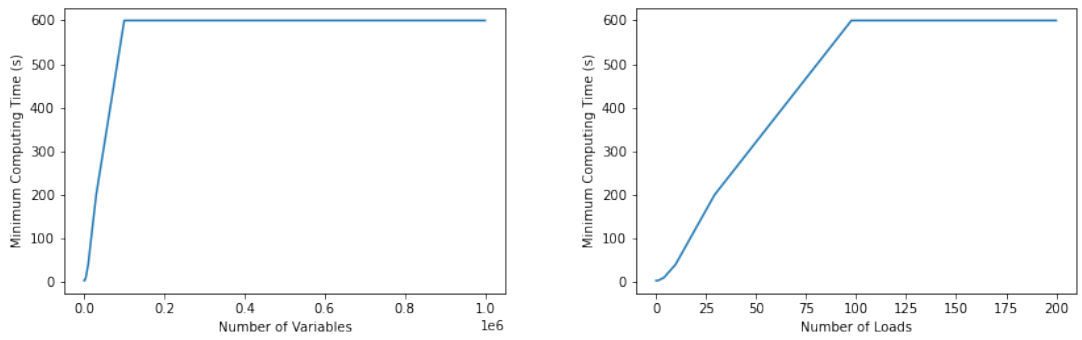
Hybrid QUBO Solver

This work implements the QUBOs for the UCP using the UQO-client. Unfortunately, the access to D-Waves hybrid samplers is limited, and this study used all the available computing time for the hybrid DQM sampler. For this reason, this work does not consider the performance of the hybrid sampler for QUBOs.

Even though this work does not run the experiments on the hybrid QUBO sampler, the minimum computing power can be computed. The hybrid QUBO sampler has a minimum computing time, just like the hybrid DQM sampler. In this case, the computing time depends on the number of variables of the QUBO. Table 5.6 lists the pairs of variable amount and computing time. Figure 5.5a shows the minimum time for all possible input sizes. Figure 5.5b shows the minimum time for the experiments represented in figure 5.3b.

Number of Variables v_{QUBO}	Minimum Computing Time t (s)
1	3.0
1024	3.0
4096	10.0
10000	40.0
30000	200.0
100000	600.0
1000000	600.0

Table 5.6: Interpolation Points for Minimum Computing Time of Hybrid DQM Solver



(a) Full

(b) Corresponding to Extended Experiment Data

Figure 5.5: Minimum Computing Time of Hybrid QUBO Solver

The minimum computing time of this sampler is greater than the computing time of the hybrid DQM sampler for large inputs. How much computing time this sampler would need additionally to the minimum computing time can't be reasoned about without actually performing experiments.

Direct QUBO Solver

D-Wave's QPUs can embed many QUBOs on them. But if a QUBO has too many variables or quadratic biases connecting the biases, D-Wave might not find an embedding. That is the case for the problems this work considers.

The smallest problem formulated as a QUBO has $1,024$ variables and $263,680$ quadratic biases. These values can be calculated using formulas (3.26a) and (3.26b) respectively. One variable can have more than 255 quadratic biases, also called a clique. The reason is that power plant 4 has 2^8 possible power levels after discretizing the power levels, and variables for the different power levels of one power plant at one time instances are connected with quadratic biases.

Since the "Advantage" QPU has a 15-way qubit connectivity, it would need an enormous amount of physical qubits to represent one logical binary variable. More as there are available on the "Advantage" QPU [D-W20a; ZBDE20]. Because of this, the minor miner can't find embeddings for the QUBOs this work considers. An algorithm designed specifically for embedding large cliques by Zbinden et al. can not embed cliques larger than 180 for the "Pegasus" architecture. That implies that it is generally very hard or even impossible to find embeddings for QUBOs with cliques larger than 255.

5.4 Performance of Gate-based Quantum Computers

This work uses the Qiskit framework by IBM for the optimization using gate-based hardware. The framework uses the algorithm described in section 2.3.2. Unfortunately, the quantum circuit is too big to be executed on any of IBM's public quantum computers for the smallest UCP that this work considers.

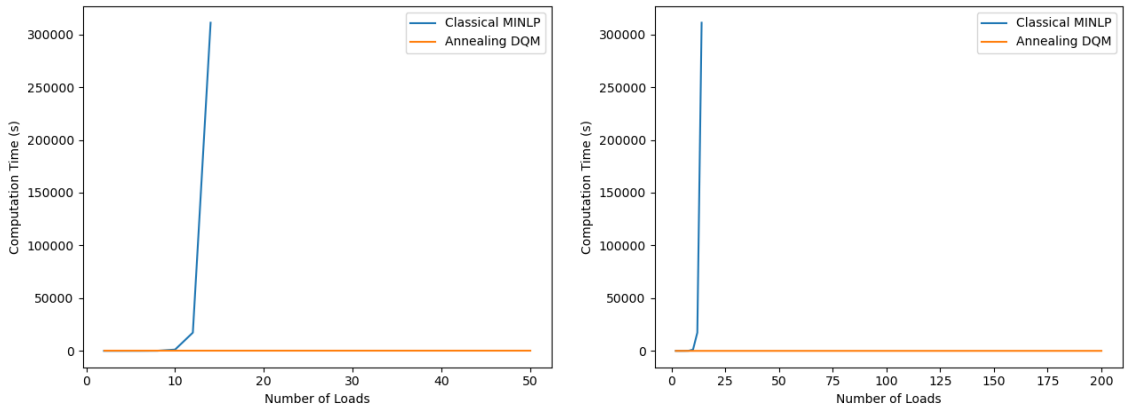
The number of qubits needed just for the QUBO variables is the same as the number of binary variables in the QUBO. Formula (3.26a) gives the number of variables in a QUBO depending on the number of time instances of the UCP. These qubits are considered the first quantum register of the quantum circuit. In the case of the smallest UCP, this work considers with the 4 power plants listed in section 5.1 and 2 time instances, the number of qubits would be $1,024$. With every additional 2 time instances of the UCP, the algorithm needs another $1,024$ qubits.

Additionally to this quantum register, the algorithm needs a second quantum register for storing the result of the QUBO dependent on the QUBO variables of the first register. The domain of values the QUBO produces grows with the size of the QUBO. It is proportional to the number of biases assuming the average value of the biases stays the same.

Regardless of the number of gates the quantum circuit would have, the gate-based quantum computers right now do not have enough qubits to handle the problems this work considers.

5.5 Comparison of Performances

The hybrid DQM sampler from D-Wave solves the UCP with 14 time instances a lot faster than the classical algorithm for solving MINLPs. For problems with more time instances, this paper has no data from the classical solver. Figure 5.6a shows the computing times needed to solve UCPs for the classical algorithm and the hybrid annealing-based algorithm for UCPs with up to 50 time instances. The line for the classical algorithm stops at 14 time instances because this work does not test the classical algorithm on bigger problems. Assuming that the classical algorithm has an exponential time complexity and the computation time continues like the graph in the figure, the hybrid annealing-based algorithm has a great advantage for bigger problems.



(a) With up to 50 Loads

(b) With up to 200 Loads

Figure 5.6: Performance Comparison of Classical and Hybrid Annealing Algorithms

Although this improved computing time is astonishing, it is worthless if the hybrid annealing-based algorithm produces non-optimal results for the UCP problems. For this reason, this work compares the solutions both algorithms produce based on the objective function ($f(x)$) (3.1a). The work computes the relative error of the solution of the hybrid annealing-based algorithm based on formula (5.3).

$$e_{\text{hybrid}}(x) = \frac{f_{\text{hybrid}}(x) - f_{\text{classical}}(x)}{f_{\text{hybrid}}(x)} \tag{5.3}$$

Figure 5.7 shows the error of the hybrid annealing-based algorithm the few experiments that this work runs using both the classical and the hybrid annealing-based algorithms. While the error stays smaller than 0.5% for these problems, it seems to grow exponentially. There is no way to know how the solution error behaves for bigger problems without running the experiments using classical hardware. Assuming that the error continues to grow exponentially with respect to the number of time instances in the UCP, the hybrid annealing-based algorithm produces worse solutions for larger UCPs. It is crucial to highlight this trade-off in this work.

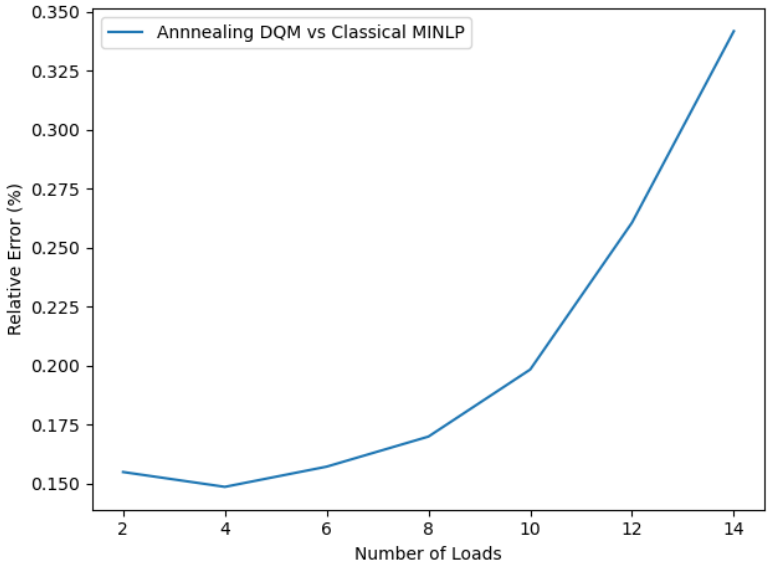


Figure 5.7: Relative Error of Hybrid Annealing Algorithm compared to Classical Algorithm

6 Conclusion

This work proposes an approach to reformulate UCPs with inter-time dependencies as problems that quantum computers can solve. DQMs and QUBOs are the names of the optimization problems that this approach produces. Gate-based quantum computers can't optimize the resulting QUBOs because they lack qubits. Annealing-based quantum computers can't optimize the resulting QUBOs directly on the QPU because they lack connections between their qubits. But hybrid approaches using both classical computing and quantum computing can optimize the resulting QUBOs and DQMs.

This work compares the performance of hybrid annealing-based optimization of DQMs with the classical optimization of MINLPs for real-world like UCPs. It is shown that the hybrid algorithm has a big computational advantage regarding the time it needs to optimize the UCP. But also, it has a computational disadvantage regarding the quality of the result. That leads to a trade-off between computation time and optimality of the result.

Future Work

The conversion of the MINLP formulation of the UCP to a DQM formulation is not optimal. While converting the UCP to a DQM, the proposed approach loses the ability to find the actual optimal solution to the UCP. This work shows this experimentally in section 5.5.

There might be a better way to formulate a DQM for the UCP. That way must conserve the ability to find the actual optimal solution to the UCP. That way could achieve the same computational advantage as demonstrated in section 5.5 if the size of the DQMs is about the same. The size of the DQM depends on the number of variables, their possible values, and the number of biases.

Another approach to mitigate the non-optimality would be taking the result and computing the details using a classical algorithm. The classical algorithm for optimizing MINLPs this work considers, "Couenne", can have a starting point for the optimization. The idea is to use the hybrid DQM sampler's result as the starting point.

Yet another promising approach is to find a hybrid algorithm for optimizing MINLP problems directly. Then the reformulation as a DQM would not be necessary. Ajagekar et al. demonstrated hybrid algorithms for similar mathematical optimization problems like the Mixed-integer Linear Problem (MILP).

Such a general-purpose hybrid MINLP solver would help not only the energy industry but also many other industries. There exists an MINLP formulation for almost every real-world industry problem [BLL+09]. Thus most real-world industry problems could be solved using such a hybrid MINLP solver.

Bibliography

[AEA09] M. R. Alrashidi, K. M. El-Naggar, A. K. Al-Othman. "Particle swarm optimization based approach for estimating the fuel-cost function parameters of thermal power plants with valve loading effects". In: *Electric Power Components and Systems* 37.11 (2009), pp. 1219–1230. ISSN: 15325008. DOI: 10.1080/15325000902993589 (cit. on pp. 49, 50).

[AHY20] A. Ajagekar, T. Humble, F. You. "Quantum computing based hybrid solution strategies for large-scale discrete-continuous optimization problems". In: *Computers and Chemical Engineering* 132 (2020). ISSN: 00981354. DOI: 10.1016/j.compchemeng.2019.106630. arXiv: 1910.13045 (cit. on pp. 15, 16, 61).

[AMJ17] S. Y. Abujarad, M. W. Mustafa, J. J. Jamian. "Recent approaches of unit commitment in the presence of intermittent renewable energy resources: A review". In: *Renewable and Sustainable Energy Reviews* 70.October 2015 (2017), pp. 215–223. ISSN: 18790690. DOI: 10.1016/j.rser.2016.11.246. URL: http://dx.doi.org/10.1016/j.rser.2016.11.246 (cit. on pp. 15, 17, 24).

[AP16] M. Aiello, G. A. Pagani. "How Energy Distribution Will Change: An ICT Perspective". In: *Smart Grids from a Global Perspective* (2016), pp. 235–248. ISSN: 1098-6596. DOI: 10.1007/978-3-319-28077-6. arXiv: arXiv:1011.1669v3. URL: http://link.springer.com/10.1007/978-3-319-28077-6 (cit. on p. 15).

[AY19] A. Ajagekar, F. You. "Quantum computing for energy systems optimization: Challenges and opportunities". In: *Energy* 179.607 (2019), pp. 76–89. ISSN: 03605442. DOI: 10.1016/j.energy.2019.04.186 (cit. on p. 24).

[Bal83] E. Balas. "Disjunctive programming". In: *Discrete Applied Mathematics* 5.2 (1983), pp. 247–248. ISSN: 0166218X. DOI: 10.1016/0166-218X(83)90046-X (cit. on p. 37).

[Bal95] R. Baldick. "The Generalized Unit Commitment Problem". In: *IEEE Transactions on Power Systems* 10.1 (1995), pp. 465–475. ISSN: 15580679. DOI: 10.1109/59.373972 (cit. on pp. 15, 17, 24).

[BAS+13] S. Boixo, T. Albash, F. M. Spedalieri, N. Chancellor, D. A. Lidar. "Experimental signature of programmable quantum annealing". In: *Nature Communications* 4.May (2013). ISSN: 20411723. DOI: 10.1038/ncomms3067. arXiv: 1212.1739 (cit. on p. 19).

[BBC+17] L. S. Bishop, S. Bravyj, A. Cross, J. M. Gambetta, S. John. "Quantum volume". In: (2017). ISSN: 17936578. DOI: 10.1142/S0217979215501660 (cit. on p. 23).

[BBHT98] M. Boyer, G. Brassard, P. HØyer, A. Tapp. "Tight bounds on quantum searching". In: *Fortschritte der Physik* 46.4-5 (1998), pp. 493–505. ISSN: 00158208. DOI: 10.1002/(SICI)1521-3978(199806)46:4/5<493::AID-PROP493>3.0.CO;2-P. arXiv: 9605034 [quant-ph] (cit. on p. 21).

[BCMR10] Z. Bian, F. Chudak, W. Macready, G. Rose. "The Ising model: teaching an old problem new tricks". In: *D-Wave Whitepaper Series* (2010), pp. 1–32. URL: https://www.dwavesys.com/sites/default/files/weightedmaxsat_v2.pdf (cit. on p. 19).

[BDB13] K. Basu, V. Debusschere, S. Bacha. "Residential appliance identification and future usage prediction from smart meter". In: *IECON Proceedings (Industrial Electronics Conference)* (2013), pp. 4994–4999. DOI: 10.1109/IECON.2013.6699944 (cit. on p. 15).

[Bie96] D. Bienstock. "Computational study of a family of mixed-integer quadratic programming problems". In: *Mathematical Programming, Series B* 74.2 PART A (1996), pp. 121–140. ISSN: 00255610. DOI: 10.1007/bf02592208 (cit. on pp. 15, 18).

[BLL+09] P. Belotti, J. Lee, L. Liberti, F. Margot, A. Wächter. "Branching and bounds tighteningtechniques for non-convex MINLP". In: *Optimization Methods and Software* 24.4-5 (2009), pp. 597–634. ISSN: 10294937. DOI: 10.1080/10556780903087124 (cit. on pp. 18, 52, 61).

[BMF20] W. Bernoudy, C. Mcgeoch, P. Farr. "D-Wave Hybrid Solver Service + Advantage: Technology Update". In: (2020). URL: https://qubits.online-event.co/mvc/view/download-file.php?type=view&id=135300 (cit. on p. 19).

[BMM+11] R. Baños, F. Manzano-Agugliaro, F. G. Montoya, C. Gil, A. Alcayde, J. Gómez. "Optimization methods applied to renewable and sustainable energy: A review". In: *Renewable and Sustainable Energy Reviews* 15.4 (2011), pp. 1753–1766. ISSN: 13640321. DOI: 10.1016/j.rser.2010.12.008 (cit. on p. 15).

[Bun] Bundesnetzagentur. *Kraftwerksliste. Download from Bundesnetzagentur webpage*. URL: https://www.bundesnetzagentur.de/DE/Sachgebiete/ElektrizitaetundGas/elektrizitaetundgas-node.html (visited on 10/15/2020) (cit. on p. 49).

[Che08] C.-l. Chen. "Optimal Wind – Thermal Generating Unit Commitment". In: *IEEE Transactions on Energy Conversion* 23.1 (2008), pp. 273–280 (cit. on p. 15).

[D-Wa] D-Wave. *DiscreteQuadraticModel Documentation*. URL: https://docs.ocean.dwavesys.com/en/stable/concepts/dqm.html (visited on 01/29/2021) (cit. on p. 39).

[D-Wb] D-Wave. *dwave-ocean-sdk Documentation*. URL: https://docs.ocean.dwavesys.com/en/stable/ (visited on 01/29/2021) (cit. on p. 39).

[D-Wc] D-Wave. *dwave-ocean-sdk GitHub Repository*. URL: https://github.com/dwavesystems/dwave-ocean-sdk (visited on 01/29/2021) (cit. on p. 39).

[D-W18] D-Wave. *System Roadmap Knoxville Overview*. Tech. rep. 2018 (cit. on p. 23).

[D-W20a] D-Wave. "Advantage Datasheet v9.0". In: (2020), pp. 1–2. URL: https://dwavesys.com/sites/default/files/Advantage_Datasheet_v9_0.pdf (cit. on pp. 19, 22, 23, 57).

[D-W20b] D-Wave. "Hybrid Solver for Discrete Quadratic Models Formulating a DQM". In: *D-Wave Whitepaper Series* (2020). URL: https://dwavesys.com/sites/default/files/14-1050A-A_Hybrid_Solver_for_Discrete_Models.pdf (cit. on pp. 20, 24, 53).

[Deu85] D. Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.11 (1985), pp. 97–117 (cit. on p. 21).

[DH96] C. Dürr, P. Høyer. "A Quantum Algorithm for finding the Minimum". In: July (1996), pp. 1–2. arXiv: 9911082 [quant-ph]. URL: http://arxiv.org/abs/quant-ph/9911082 (cit. on p. 15).

[Foua] C.-O. Foundation. *COIN-OR Home Page*. URL: https://coin-or.org (visited on 12/05/2020) (cit. on p. 52).

[Foub] C.-O. Foundation. *Couenne GitHub Repository*. URL: https://github.com/coin-or/Couenne (visited on 12/05/2020) (cit. on p. 52).

[Gam] J. Gambetta. *IBM Quantum Roadmap*. URL: https://www.ibm.com/blogs/research/2020/09/ibm-quantum-roadmap/ (visited on 02/27/2021) (cit. on pp. 22, 23).

[GDP+12] I. Georgievski, V. Degeler, G. A. Pagani, T. A. Nguyen, A. Lazovik, M. Aiello. "Optimizing energy costs for offices connected to the smart grid". In: *IEEE Transactions on Smart Grid* 3.4 (2012), pp. 2273–2285. ISSN: 19493053. DOI: 10.1109/TSG.2012.2218666 (cit. on p. 49).

[GM] J. Gambetta, D. McClure. *Hitting a Quantum Volume Chord: IBM Quantum adds six new systems with Quantum Volume 32*. URL: https://www.ibm.com/blogs/research/2020/07/qv32-performance/ (visited on 02/27/2021) (cit. on p. 23).

[Gro96] L. K. Grover. "A fast quantum mechanical algorithm for database search". In: *European Physical Journal C* 75.6 (1996), pp. 212–219. ISSN: 14346052. DOI: 10.1140/epjc/s10052-015-3475-9. arXiv: 1504.00611 (cit. on p. 21).

[GWG19] A. Gilliam, S. Woerner, C. Gonciulea. "Grover adaptive search for constrained polynomial binary optimization". In: *arXiv* (2019), pp. 1–8. ISSN: 23318422. arXiv: 1912.04088 (cit. on pp. 15, 22).

[HLR] HLRS. *HLRS Website*. URL: https://www.hlrs.de/home/ (visited on 01/07/2021) (cit. on p. 52).

[HWW11] W. E. Hart, J.-P. Watson, D. L. Woodruff. "Pyomo: modeling and solving mathematical programs in Python". In: *Mathematical Programming Computation* 3.3 (2011), pp. 219–260 (cit. on p. 37).

[KBL+12] N. Kumar, P. Besuner, S. Lefton, D. Agan, D. Hilleman. *Power Plant Cycling Costs*. Tech. rep. 2012 (cit. on pp. 49, 50).

[Lab] S. N. Laboratories. *Solving Pyomo Models*. URL: https://pyomo.readthedocs.io/en/stable/solving_pyomo_models.html (visited on 12/05/2020) (cit. on p. 52).

[LB20] F. Leymann, J. Barzen. *The bitter truth about gate-based quantum algorithms in the NISQ era*. Vol. 5. 4. 2020, pp. 1–42. ISBN: 0000000183. DOI: 10.1088/2058-9565/abae7d. arXiv: arXiv:2006.02856v1 (cit. on p. 15).

[LLL19] Z. J. Lee, T. Li, S. H. Low. "ACN-Data: Analysis and applications of an open EV charging dataset". In: *e-Energy 2019 - Proceedings of the 10th ACM International Conference on Future Energy Systems* (2019), pp. 139–149. DOI: 10.1145/3307772.3328313 (cit. on p. 49).

[LMa] Q. A. R. Laboratory, L. M. U. Munich. *UQO PyPI page*. URL: https://pypi.org/project/uqo/ (visited on 03/10/2021) (cit. on p. 42).

[LMb] Q. A. R. Laboratory, L. M. U. Munich. *UQO GitHub repository*. URL: https://github.com/QAR-Lab/uqoclient (visited on 03/10/2021) (cit. on p. 42).

[LS05] T. Li, M. Shahidehpour. "Price-Based Unit Commitment : A Case of Lagrangian Relaxation Versus Mixed Integer Programming". In: *IEEE Trans. on Power Systems* 20.4 (2005), pp. 2015–2025 (cit. on p. 15).

[PH00] D. Portnov, T. Hogg. "Quantum Optimization". In: *Proceedings of the Joint Conference on Information Sciences* 5.1 (2000), pp. 778–781. arXiv: 0006090 [quant-ph] (cit. on p. 15).

[Pro] E. Project. *Excess Project Website*. URL: http://excess-project.eu/software.html (visited on 01/07/2021) (cit. on p. 52).

[Qisa] Qiskit. *Qiskit GitHub repository*. URL: https://github.com/Qiskit (visited on 03/12/2021) (cit. on p. 45).

[Qisb] Qiskit. *Qiskit website*. URL: https://qiskit.org/ (visited on 03/12/2021) (cit. on p. 45).

[Sho98] P. Shor. "Quantum Computing". In: *International Congress of Mathematicians* I (1998), pp. 467–486 (cit. on p. 21).

[SKMM13] A. Schröder, F. Kunz, J. Meiss, R. Mendelevich. *Current and prospective costs of electricity generation until 2050*. Tech. rep. 2013. URL: http://hdl.handle.net/10419/80348 (cit. on pp. 49, 50).

[SUN+19] R. Shaydulin, H. Ushijima-Mwesigwa, C. F. Negre, I. Safro, S. M. Mniszewski, Y. Alexeev. "A hybrid approach for solving optimization problems on small quantum computers". In: *Computer* 52.6 (2019), pp. 18–26. ISSN: 15580814. DOI: 10.1109/MC.2019.2908942 (cit. on pp. 15, 16).

[Tec] C. I. of Technology. *Download from ACN webpage. Site: Caltech, From: 2020-01-01, To: 2020-10-11, Minimum Energy: 0 kWh*. URL: https://ev.caltech.edu/dataset (visited on 10/11/2020) (cit. on p. 49).

[VØ09] J. G. Vlachogiannis, J. Østergaard. "Reactive power and voltage control based on general quantum genetic algorithms". In: *Expert Systems with Applications* 36.3 PART 2 (2009), pp. 6118–6126. ISSN: 09574174. DOI: 10.1016/j.eswa.2008.07.070. URL: http://dx.doi.org/10.1016/j.eswa.2008.07.070 (cit. on p. 15).

[VP98] V. Vedral, M. B. Plenio. "Basics of quantum computation". In: *Progress in Quantum Electronics* 22.1 (1998), pp. 1–39. ISSN: 00796727. DOI: 10.1016/S0079-6727(98)00004-4. arXiv: 9802065 [quant-ph] (cit. on pp. 18, 19).

[ZBDE20] S. Zbinden, A. Bärtschi, H. Djidjev, S. Eidenbenz. *Embedding algorithms for quantum annealers with chimera and pegasus connection topologies*. Vol. 12151 LNCS. Springer International Publishing, 2020, pp. 187–206. ISBN: 9783030507428. DOI: 10.1007/978-3-030-50743-5_10. URL: http://dx.doi.org/10.1007/978-3-030-50743-5_10 (cit. on pp. 22, 57).

[ZHD+16] N. Zhang, Z. Hu, D. Dai, S. Dang, M. Yao, Y. Zhou. "Unit commitment model in smart grid environment considering carbon emissions trading". In: *IEEE Transactions on Smart Grid* 7.1 (2016), pp. 420–427. ISSN: 19493053. DOI: 10.1109/TSG.2015.2401337 (cit. on pp. 15, 19).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Urbach, 12.04.2021, FLWeik

place, date, signature