Quantum-Classical Optimization in Machine Learning

Dissertation

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von

Sascha Mücke

Dortmund

2025

Tag der mündlichen Prüfung:	13. Mai 2025
Dekan:	Prof. Dr. Jens Teubner
Gutachter*innen:	Prof. Dr.-Ing. Petra Wiederkehr
	Prof. Dr. Christian Bauckhage

Abstract

Machine Learning (ML) is a driving force of innovation and a key technology of the future. At the same time, Quantum Computing (QC) is emerging as a technology that holds the potential of asymptotic speedups and efficient computation in exponentially large spaces. It also enables faster optimization, which lies at the core of ML, leading to ongoing efforts of utilizing QC to perform ML. However, QC is still in its infancy, and applications of quantum-based optimization and ML are severly limited by imperfect hardware. This thesis explores strategies of using QC to enhance ML on the one hand, and using classical optimization to enhance QC in its current restricted state on the other. To this end, a feature selection method based on a Qubo embedding is discussed, which is deployed on a quantum annealer. For Support Vector Machines, a classical ML model, two embeddings on quantum computers are shown, using both paradigms of adiabatic QC and gate-based QC. It is shown how evolutionary optimization can be used to jointly learn the structure and parameters of quantum circuits. Further, it is shown that low precision of Qubo weights can lead to loss of performance on quantum annealers, and strategies to mitigate this effect are presented, leading to higher-quality optimization results. Taken together, this thesis broadens the scope of quantum-classical computation by both adding to the toolkit of quantum-enhanced ML methods, and by improving the quality of near-term QC itself. Finally, this thesis demonstrates in a range of practical applications how quantum-classical optimization can be applied in a resource-aware fashion, employing various techniques to utilize near-term QC effectively.

Acknowledgments

Ich danke meinen Freunden und Kollegen, die mich auf dem Weg begleitet haben, der zu dieser Arbeit geführt hat, allen voran Matthias Jakobs, Raphael Fischer und Ann-Kathrin Oster, die mir immerzu mit Rat und Tat zur Seite standen. Daneben danke ich Dr. Nico Piatkowski und Thore Gerlach, die gemeinsam mit den zuvor Genannten in Zeiten, da man selten das Haus verlassen konnte, die Eintönigkeit im Home-Office vertrieben haben und immer für Gespräche offen waren. Nico danke ich besonders für seine Unterstützung und Betreuung, die mich fachlich sehr vorangebracht und geprägt hat.

Ich danke meinen Betreuerinnen und Betreuern, die mich in den vergangenen Jahren begleitet und unterstützt haben. Dr. Katharina Morik danke ich für ihre Arbeit an ML2R, dem Lamarr-Institut und darüber hinaus, die meine Arbeit in der Wissenschaft erst ermöglicht hat. Weiterhin danke ich Dr. Petra Wiederkehr und Dr. Christian Bauckhage für ihre Unterstützung, auch unter ungewöhnlichen Umständen.

Letztlich danke ich auch meiner Familie und meinen Freunden, die zu allen Zeiten immer für mich da waren: Meinen Eltern, Großeltern, meiner Schwester, und allen, die mich auf meinem Weg bestärkt und ermutigt haben.

I would like to thank my friends and colleagues who have accompanied me on the path that has led to this work, above all Matthias Jakobs, Raphael Fischer and Ann-Kathrin Oster, who have always been at my side with help and advice. I would also like to thank Dr. Nico Piatkowski and Thore Gerlach, who, together with the aforementioned, have dispelled the monotony of working from home in times when I could rarely leave the house, and were always open for discussions. I would especially like to thank Nico for his support and mentoring, which really helped and shaped me professionally.

I would also like to thank my supervisors who have accompanied and supported me over the years. I would like to thank Dr. Katharina Morik for her work at ML2R, the Lamarr Institute and beyond, which made my work in science possible in the first place. I would also like to thank Dr. Petra Wiederkehr and Dr. Christian Bauckhage for their support, even under unusual circumstances.

Finally, I would also like to thank my family and friends, who have always been there for me: My parents, grandparents, my sister, and everyone who has encouraged and supported me along the way.

Acronyms

ADMM	Alternating Direction Method of Multipliers.
ANN	Artificial Neural Network.
AQC	Adiabatic Quantum Computing.
CNN	Convolutional Neural Network.
DR	Dynamic Range.
DT	Decision Tree.
EA	Evolutionary Algorithm.
ECL	Evolutionary Circuit Learning.
ET	Extra Tree.
FPGA	Field-Programmable Gate Array.
FS	Feature Selection.
GD	Gradient Descent.
GQC	Gate-based Quantum Computing.
ICE	Integrated Control Error.
LR	Logistic Regression.
MI	Mutual Information.
ML	Machine Learning.
MSE	Mean Squared Error.
NEAT	Neuro-Evolution of Augmenting Topologies.
NISQ	Noisy Intermediate-Scale Quantum.
PCA	Principal Component Analysis.
QA	Quantum Annealing.
QC	Quantum Computing.
QFS	Qubo-based Feature Selection.

QML	Quantum Machine Learning.
Qubo	Quadratic Unconstrained Binary Optimization.
ReLU	Rectified Linear Unit.
RF	Random Forest.
RFE	Recursive Feature Elimination.
RQSVM	Real-part Quantum Support Vector Machine.
RQSVR	Real-part Quantum Support Vector Regressor.
SA	Simulated Annealing.
SGD	Stochastic Gradient Descent.
SVM	Support Vector Machine.
SVR	Support Vector Regressor.
VQC	Variational Quantum Circuit.
VQE	Variational Quantum Eigensolver.

Contents

I. Fundamentals 1

1. Introduction 3
   1.1. Outline 6
   1.2. Overview of Publications 8
      1.2.1. Author Attribution 9
      1.2.2. Source Attribution 10

2. Background 11
   2.1. Notational Conventions 11
      2.1.1. Binary Vectors 13
   2.2. Optimization 13
      2.2.1. Gradients in Optimization 14
      2.2.2. Evolutionary Optimization 15
      2.2.3. Quadratic Unconstrained Binary Optimization 16
   2.3. Machine Learning 20
      2.3.1. Feature Selection 25
      2.3.2. Support Vector Machines 26
   2.4. Quantum Computing 28
      2.4.1. The Circuit Model 31
      2.4.2. Variational Quantum Circuits 32
      2.4.3. Adiabatic Quantum Computing 34
      2.4.4. The NISQ Era 36
      2.4.5. Working with Quantum Hardware 37
      2.4.6. State of the Art 38

II. Quantum Computing for Classical Machine Learning 41

3. Feature Selection on Quantum Annealers 43
   3.1. QUBO-Based Feature Selection 44
   3.2. Feature Subset Size 46
   3.3. The QFS Algorithm 49
      3.3.1. Overview of Similar Approaches 50
   3.4. Experimental Evaluation 51
      3.4.1. Feature Subset Quality 52
      3.4.2. Cross-Model Comparison 53

3.4.3. Lossy Compression with an Autoencoder 55
3.4.4. QFS on Quantum Hardware 58
3.5. Concluding Remarks 60

4. Quantum Support Vector Machines 65
4.1. SVM Training through Quantum Annealing 66
4.1.1. Increasing Precision 67
4.1.2. Experimental Evaluation 68
4.1.3. Concluding Remarks 70
4.2. Gate-Based Support Vector Machine (SVM) Deployment 70
4.2.1. Evaluating the Performance of Quantum Inference 74
4.2.2. Increasing the Number of Features 76
4.2.3. Comparison to a Conventional Quantum SVM 77
4.3. Concluding Remarks 79

III. Classical Algorithms for Quantum Computing 81

5. Quantum Circuit Evolution 83
5.1. Evolutionary Circuit Learning 86
5.1.1. A Universal Gate Set 87
5.1.2. Mutating Quantum Circuits 88
5.2. Experimental Evaluation 90
5.2.1. Target Hamiltonians 90
5.2.2. Comparing Quantum Circuit Evaluations 95
5.3. Concluding Remarks 96

6. Parameter Compression for Adiabatic Quantum Computing 97
6.1. Preserving Optima 99
6.2. The Role of Dynamic Range 100
6.3. An Optimal Rounding Strategy 102
6.4. QUBO Parameter Compression 104
6.4.1. Bounding the Minimal Energy 104
6.4.2. Reducing the Dynamic Range 107
6.5. Heuristic Compression Strategies 109
6.5.1. Choosing the Next Parameter 112
6.6. Parameter Compression in Practice 112
6.6.1. Results for Random Instances 113
6.6.2. Results on Quantum Hardware 116
6.7. Concluding Remarks 121

IV. Applications of Quantum-Classical Algorithms 123

7. Optimal Light-Source Placement 125
7.1. The TorchPlacement Problem . . . 127
7.2. A Quadratic Unconstrained Binary Optimization (Qubo) formulation of TorchPlacement . . . 128
7.2.1. Handling Inequality Constraints . . . 130
7.3. Experimental Evaluation . . . 132
7.4. Concluding Remarks . . . 134

8. Solving Sudoku using Quantum Annealing 137
8.1. The Sudoku Qubo . . . 139
8.2. Incorporating Clues . . . 140
8.2.1. Saving Quantum Resources through Clamping . . . 141
8.3. A Practical Example . . . 142
8.4. Effect of Puzzle Difficulty . . . 144
8.5. Sampling Sudoku Puzzles from Quantum States . . . 145
8.5.1. Generating Unambiguous Puzzles . . . 146
8.6. Concluding Remarks . . . 146

9. Predicting Process Stability in Machining 149
9.1. Stability of Machining Processes . . . 150
9.1.1. Data Set of Stability Limits . . . 150
9.2. Applying Quantum Support Vector Regression . . . 151
9.2.1. Building a Feature Map . . . 152
9.2.2. Support Vector Regression . . . 153
9.3. Evaluation on Stability Data . . . 154
9.3.1. Predicting Tools . . . 155
9.4. Concluding Remarks . . . 158

10. Conclusion and Future Work 159

Bibliography 163

Part I.

Fundamentals

1. Introduction

Optimization lies at the core of Machine Learning (ML), which has been a driving force of innovation over the past few decades. ML models are used to learn from example data with the goal of performing tasks usually associated with (human) intelligence: Object recognition in images, classification and regression of tabular data, time series analysis and forecasting, language translation and generation, and much more is today not only feasible, but widely available and readily applicable to a plethora of application areas. Consequently, ML has attained a firm place in the public conscience as a key technology of the future.

All models have in common that they are trained through some type of optimization routine that minimizes a loss function quantifying the model's performance on previously unobserved data. Many of the optimization problems occurring in ML are very challenging, particularly because they are high-dimensional, involving millions or even billions of parameters trained on millions of data points. For most deep learning models, Stochastic Gradient Descent (SGD) is the optimization method of choice, where the gradient of the loss function w.r.t. small batches of data points is computed using backpropagation [9]. Obstacles like the vanishing gradient problem [10] have been overcome using carefully designed activation functions [11], and special-purpose hardware that massively parallelizes linear algebra operations has drastically increased the scale at which deep learning can be applied, most recently enabling large language models such as GPT and its successors [12]. Training such massive models is, however, becoming increasingly costly and inefficient, requiring ever more time and compute resources to traverse the exponentially growing search space of parameters, and contributing to climate change along the way [13, 14]. Moreover, gradient methods are generally bound to find only local optima, and when the solution candidate space of a problem scales exponentially with the problem size, finding the exact solution quickly becomes infeasible. For many loss functions, a gradient cannot be computed or does not exist at all, e.g., when dealing with discrete search spaces.

In a parallel development, Quantum Computing (QC) is slowly evolving from a purely theoretical discipline into a practically applicable technology. At its heart lies the prospect of asymptotic speed-ups that enable efficient optimization and bigger, richer computations in short time spans, for which *classical* (that is, non-quantum) methods would need centuries or more. Using quantum circuits, algorithms can be built and executed, some of which famously asymptotically outperform classical algorithms: Probably the most famous example is Shor's Algorithm, which solves the problem of prime factorization of a natural number $n \in \mathbb{N}$ using $\mathcal{O}((\log n)^2)$ gates [15], posing a (theoretical) threat to cryptography, which largely relies on the hardness of decomposing large prime numbers. Another example is Grover Search, which can be used to search lists of length n in $\mathcal{O}(\sqrt{n})$ time [16]. Quantum annealers have become a novel target platform for quantum-accelerated optimization [17]. Drivers of this development are improvements in QC hardware, fueled by the continuous improvement of superconducting technology like Josephson junctions [18] and the transmon qubit [19]. Companies like IBM and D-Wave make their quantum hardware and learning resources available to the public, making QC accessible and thus fostering new ideas and research.

The central concept of QC is replacing bits used in classical computing with *qubits* (contraction of "quantum bits"), which, just like normal bits, have two basis states, $|0\rangle$ and $|1\rangle$. However, they can be in *superposition*, which means they assume one or the other state with a certain probability when being measured. Therefore, measuring an isolated qubit can be thought of as taking samples from a Bernoulli-distributed random variable. Groups of qubits can be *entangled*, meaning that their marginal measurement probabilities are not mutually independent. The joint states of such *systems* of qubits can be expressed by exponentially large complex-valued vectors. However, these states can be manipulated through physical operations without the need to instatiate the vectors in any sort of memory. The central question of QC is how to exploit this implicit manipulation of exponentially large state spaces in order to do something meaningful.

There are two major paradigms that QC research follows: Gate-based Quantum Computing (GQC) and Adiabatic Quantum Computing (AQC). GQC models quantum state manipulations as sequences of simple operations on single qubits or pairs of qubits. In analogy to logical circuits, these sequences are called *quantum circuits*, and the individual operations *gates*, some of which take real-valued parameters. If a set of gates can be proven to be universal, every possible quantum state can be constructed from a (possibly very long) sequence of gates exclusively from this set, similar to the instruction set of a classical processor. The resulting quantum state can be measured, yielding an empirical distribution over binary vectors, which in turn can be interpreted according to the application at hand.

The second paradigm, AQC, follows a different approach by encoding optimization problems into time-dependent Hamiltonian operators, which can be thought of as functions assigning an energy to every quantum state. Their *ground state* (i.e., the quantum state of lowest energy) encodes the optimal solution. Special-purpose hardware has been developed since the 2000s that performs Quantum Annealing (QA), a hybrid form of AQC that can be executed on the imperfect quantum hardware that exists today. Here, problems are encoded into a parametrized Hamiltonian (usually of an Ising model), and a quantum state prepared in the known ground state of a simple Hamiltonian slowly evolves toward the target Hamiltonian. If done slowly enough, the system remains in its ground state, and the optimal solution of the target problem can simply be measured at the end. The class of problems that can be solved this way is called QUBO, which is NP-hard and has a wide range of applications across many disciplines, ranging from its origins in economics to resource allocation and logistics, satisfiability and graph problems (see Section 2.2.3).

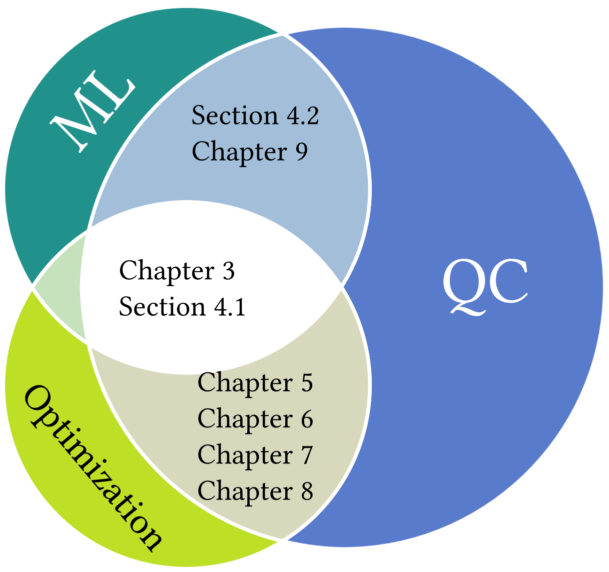


Figure 1.1.: Intersections of the topics of ML, Optimization, and QC, and where they are discussed in this thesis.

Despite its favorable theoretical properties and the highly promising developments over the last decades, quantum hardware has not yet reached a state where it can be widely applied to large-scale problems in practice and yield state-of-the-art results that surpass high-performance classical computing devices. The current state of quantum hardware has been dubbed the Noisy Intermediate-Scale Quantum (NISQ) era, as both the number of usable qubits as well as the number of gates that can be applied in a quantum circuit are severely limited due to effects such as decoherence and gate noise [20]. Still, in its ambitious roadmap, IBM expects fully error-corrected quantum computers with several thousand qubits from 2030 onwards[1]. In this light, new research directions have emerged, attempting to make efficient use of the currently available quantum hardware with the aim of supplementing or completely replacing classical methods found throughout computer science, expecting them to one day be used on a large scale on noise-free quantum computers.

As a notable example of such an emerging research direction, Quantum Machine Learning (QML) attempts to replace models, algorithms, and routines used in ML with new quantum counterparts [21]. The approaches found in literature are largely heuristic. For instance, parametrized quantum circuits that structurally imitate Artificial Neural Networks (ANNs) are trained by encoding input data into the gate parameters, measuring the resulting state, interpreting it as a prediction (e.g., a one-hot encoding), and computing parameter updates based on a loss value. However, such models have their own challenges in training [22, 23], and a clear advantage over classical ML has not yet been

[1]`https://www.ibm.com/roadmaps/quantum/`

observed [24]. In fact, QC constitutes a novel computing paradigm whose true native use case may yet be discovered. As there is "no free lunch" in supervised learning [25], it is easily conceivable that there are (learning) tasks that are perfectly suited for quantum models. Clearly, different applications require different hardware architectures: Applications with very high data throughput may benefit from Field-Programmable Gate Arrays (FPGAs), which achieve much higher clock speeds and thus can handle real-time requirements [26, 27]. Other application areas, such as the "Internet of Things", have very strict limits on size and power consumption, for which microcontrollers are a great option [28]. Therefore, expanding our collection of available quantum algorithms and their applications is a promising endeavor. At the same time, paving the path out of the Noisy Intermediate-Scale Quantum (NISQ) era—or, at least, mitigating its current limitations—is another necessary research direction: By allowing for more effective exploration of QC's current capabilities and, consequently, faster development of future-proof QC algorithms, applying our vast toolbox of theoretical knowledge and classical algorithms to try and improve the efficacy of today's quantum hardware facilitates research both in the short and the long term.

This thesis takes a step in both directions by exploring some of the various ways in which *(i)* QC in its NISQ state can supplement ML tasks, and *(ii)* classical algorithms can supplement current NISQ-era quantum computers. In this way, we add to a new toolbox of methods that can be used in quantum-enhanced ML settings. In particular, we show how QA can be used to perform Feature Selection, and how Support Vector Machines can be trained and deployed using both QC paradigms. Conversely, we improve upon the usability of quantum hardware in two ways: We use Evolutionary Algorithms (EAs) to construct quantum circuits from scratch without any pre-defined *ansatz*. Moreover, we show that the solution quality of QA is dependent on the dynamic range of the Qubo problem's parameters, and devise strategies to reduce it, yielding consistently better results on QA hardware. In addition, this thesis showcases a number of applications combining ML, Optimization, and QC, highlighting various methods of approaching practical problems while maintaining feasibility on NISQ devices. The following section gives a more detailed overview of how the remainder of this thesis is structured.

1.1. Outline

This thesis is split into four parts: Part I contains the introduction and background chapters. Part II focuses on how QC can be used to supplement ML methods, while Part III covers the opposite direction, answering the question of how classical methods can support quantum computing technologies. Finally, Part IV presents three applications that combine Optimization, ML, and QC, illustrating their interplay more concretely, before this thesis concludes with a discussion about the future developments of QC and its possible implications for Optimization and ML. Figure 1.1 shows at a glance how each chapter falls into the intersections of the three pillar topics. The following paragraphs give a more detailed overview of the contents of each part.

Part I introduces the concepts of QC, Optimization, and ML. Chapter 1 gives the motivation for studying the interplay of these three topics, highlighting the current challenges and prospects of QC. Chapter 2 lays the foundation for all the following chapters by explaining the various concepts needed to follow this thesis. After establishing some notational conventions and mathematical tools in Section 2.1, the topics of optimization (Section 2.2), ML (Section 2.3), and QC (Section 2.4) are presented from the ground up. In Section 2.3, particular focus is placed on Feature Selection (Section 2.3.1), an important pre-processing step in ML pipelines, as well as the Support Vector Machine model (Section 2.3.2). These form the basis of QC methods presented in Chapters 3 and 4, respectively.

Part II dives into the core question of this thesis, presenting two approaches to implementing QC into ML methods: Chapter 3 explains how Feature Selection (FS) can be formulated as a QUBO problem, which is then solved on a quantum annealer. In a series of experiments, we show that this novel quantum-enhanced FS strategy called QUBO-based Feature Selection (QFS) produces useful feature subsets and is competitive with other FS methods. In addition, we prove based on the energy function of the QUBO problem that QFS is capable of finding any desired number k of features. Chapter 4 focuses on the SVM as a time-tested ML classification model with strong theoretical foundations. We show in Section 4.1 how support vectors can be found with a quantum annealer, using a slightly simplified formulation of the SVM optimization problem. In Section 4.2, we show how a pre-trained SVM can be deployed as a quantum circuit, utilizing a gate-based quantum computer as a sampler to compute the model prediction.

Part III takes the mirror-image approach of Part II by answering the question of how classical optimization methods can help quantum computers in their current imperfect state. To this end, Chapter 5 demonstrates how Evolutionary Optimization can be used to construct quantum circuits given a target Hamiltonian. This optimization strategy is not gradient-based and is therefore able to optimize over the discrete set of possible quantum circuits. In contrast to fixed ansätze, which are typically heuristic and overly general, we show that using a “bottom-up” approach, i.e., constructing circuits from scratch, leads to more compact circuit designs. Finally, Chapter 6 again focuses on QA, demonstrating the fundamental connection between the dynamic range of QUBO parameters and the resulting solution quality obtained from quantum annealers: A high dynamic range leads to an increased proportion of non-optimal solutions in the solution sample set. To mitigate this effect, we develop an algorithm to reduce the dynamic range while preserving the optimal solution of the QUBO problem through theoretical bounds on the optimal energy.

Part IV highlights three real-world applications that combine Optimization and ML with QC: In Chapter 7, the problem of efficient placement of light sources on a given

3D map is solved using a combination of QA and classical constraint optimization techniques, demonstrating how QC can be used as a component within a larger optimization algorithm. Chapter 8 uses the game of Sudoku as a toy example of an NP-hard optimization problem that is solved using QA: The focus lies on the technique of clamping, which allows for a substantial subset of binary variables to be fixed to constant values and removed from the optimization problem entirely, reducing the overall problem size. This technique illustrates how optimization problems can be formulated in a way that caters to the limited NISQ annealers available today. The final chapter of this part, Chapter 9, tackles the problem of stability prediction in milling processes. This regression problem, which is of great importance for obtaining high-quality results in machining applications, is solved using a modified version of the GQC-based SVM model presented in Section 4.2, fed with custom features derived from a real-world data set of empirical milling stability data. Finally, Chapter 10 concludes this thesis and gives an outlook on possible developments in QC and its interplay with Optimization and ML in the future.

1.2. Overview of Publications

This thesis covers a number of scientific papers that were written between 2019 and 2024. Parts of these papers were funded by the Federal Ministry of Education and Research of Germany (BMBF), initially as part of the Competence Center for Machine Learning Rhine-Ruhr (ML2R), later as part of the Lamarr Institute for Machine Learning and Artificial Intelligence.

Peer-Reviewed Publications

[1] Sascha Mücke, Thore Gerlach and Nico Piatkowski. “Optimum-Preserving QUBO Parameter Compression”. In: *Quantum Machine Intelligence* 7 (2025). DOI: 10.1007/s42484-024-00219-3.

[2] Nico Piatkowski and Sascha Mücke. “Real-Part Quantum Support Vector Machines”. In: *Proceedings of Machine Learning and Knowledge Discovery in Databases.* Vol. 14948. Springer, 2024. pp. 144–160. DOI: 10.1007/978-3-031-70371-3_9.

[3] Sascha Mücke. “A Simple QUBO Formulation of Sudoku”. In: *Companion Proceedings of the Genetic and Evolutionary Computation Conference.* ACM, 2024, pp. 1958–1962. DOI: 10.1145/3638530.3664106.

[4] Sascha Mücke and Thore Gerlach. “Efficient Light Source Placement using Quantum Computing”. In: *Proceedings of the Conference on “Lernen, Wissen, Daten, Analysen”.* Vol. 3630. 2023, pp. 478–491.

[5] Sascha Mücke, Raoul Heese, Sabine Müller, Moritz Wolter and Nico Piatkowski. “Feature Selection on Quantum Computers”. In: *Quantum Machine Intelligence* 5 (2023). DOI: 10.1007/S42484-023-00099-Z.

[6] Lukas Franken, Bogdan Georgiev, Sascha Mücke, Moritz Wolter, Raoul Heese, Christian Bauckhage and Nico Piatkowski. "Quantum Circuit Evolution on NISQ Devices". In: *Proceedings of the IEEE Congress on Evolutionary Computation,* 2022, pp. 1–8. DOI: `10.1109/CEC55065.2022.9870269`.

[7] Sascha Mücke, Nico Piatkowski and Katharina Morik. "Learning Bit by Bit: Extracting the Essence of Machine Learning". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen".* Vol. 2454. 2019, pp. 144–155.

Non-Peer-Reviewed Publications

[8] Sascha Mücke, Felix Finkeldey, Nico Piatkowski, Tobias Siebrecht, Petra Wiederkehr. "Predicting Machining Stability with a Quantum Regression Model". 2024. arXiv: `2412.04048 [quant-ph]`.

1.2.1. Author Attribution

The publications this thesis builds upon have been developed together with numerous co-authors. The following gives a detailed breakdown of every author's contributions. Authors are enumerated in the order of their appearance on the author list of the publication in question. The author of this thesis is highlighted in boldface.

For [1], **author 1** developed the concept of Qubo parameter compression, devised and proved all theoretical insights, implemented and conducted all experiments and created their result plots, except for those investigating the DR ratio, state ordering, and unique parameter ratio on random Qubo instances, and wrote the majority of the article. Authors **1** and 2 jointly developed and formalized the heuristic compression strategies, whose Python implementation was contributed by author 2. Author 2 conducted experiments on random Qubo instances and plotted their results. They further wrote sections about the heuristic compression strategies and discussion of experimental results. Author 3 provided additional advice and did proof-reading.

For [2], authors 1 and **2** jointly developed the concept of the RQSVM model. Author 1 formalized the theoretical insights and wrote most of the paper. **Author 2** implemented the model along with a custom quantum simulator, conducted the simulated experiments, created their result plots, and wrote sections about the experiments in the paper. Author 1 conducted and documented experiments on real quantum hardware.

For [4], **author 1** developed the idea of the TorchPlacement problem, formalized it, devised the basic Qubo formulation, established the connection to SetCover, and wrote the majority of the paper. They also implemented the random generation and graphical representation of height maps. Authors **1** and 2 jointly implemented and conducted the experiments. Author 2 had the idea to use ADMM, contributed a Python implementation, and wrote the corresponding paper sections.

For [5], **author 1** developed the idea of using mutual information to perform feature selection using Qubo. He devised the Qubo formulation, the method to control the feature subset size and its correctness proof, and the final QFS algorithm. Further, he designed and conducted all experiments and wrote the majority of the paper. Author 2 assisted in experiment design and evaluation, providing result plots for all experiments. Authors 2, 3, 4, and 5 provided valuable feedback in discussions, helping to design the experiments and structure the paper. Authors 2 and 5, in particular, gave useful advice for formalizing the proof. Author 5 contributed the formal description of mutual information on discrete random variables. All authors helped in proof-reading.

For [6], authors 1 and 2 had the initial idea of training quantum circuits to minimize the expectation w.r.t. some Hamiltonian and wrote sections about quantum computing and Hamiltonians in the paper. **Author 3** determined the focus of this paper by introducing the idea of using evolutionary optimization for circuit learning. He defined the mutation operator and implemented the optimization routine in Python. Further, he formalized the the evolutionary algorithm and wrote the corresponding sections in the paper. Experiments were designed and implemented jointly by authors 1 and **3**. **Author 3** implemented software tools for the evaluation of experimental results, providing data for Fig. 5.5. Authors **3** and 7 created the plot in Fig. 5.8, all other plots were created by authors 1 and 5. All authors contributed ideas about the experiment design and structure of the paper in discussions, and did proof-reading of the paper. Author 6 provided supervision.

For [7], **author 1** devised the strategies to encode SVM and MRF in Qubo, wrote the paper, and created all figures. Author 2 contributed ideas and helped structure the paper. Author 3 provided proof-reading and supervision.

For [8], **author 1** had the initial idea of using quantum computing in a machining context, which was developed further in discussions with authors 2, 3, 4 and 5. The data was provided by authors 2, 4, and 5. The experiments were designed, implemented and conducted by **author 1**, who also created the result plots and wrote most of the paper. Author 2 contributed background sections about machining, stability in milling processes, and the data collection process. Authors 2 and 5 did proof-reading, and author 5 provided supervision.

1.2.2. Source Attribution

A detailed breakdown of which parts of the aforementioned publications were used is given in a gray box at the beginning of each chapter. Image sources are stated in the caption. Sources of definitions, theorems, and propositions are stated in their titles. If no source is given, the image, text or equation was created by the author of this thesis.

2. Background

This chapter establishes some notation in Section 2.1 that is going to be used throughout this thesis, and introduces basic concepts that the remaining chapters build upon. In particular, it introduces optimization (Section 2.2) and ML (Section 2.3), focusing on the specific problems and models that are relevant for later chapters, and gives an overview of quantum computing in its two main paradigms of GQC (quantum circuits, Section 2.4.1) and AQC (including quantum annealing, Section 2.4.3). Further, the connection between the QUBO problem (see Section 2.2.3) and its physical counterpart, the Ising model, is established, giving rise to the concept of AQC and serving as a bridge between classical and quantum optimization.

2.1. Notational Conventions

Throughout this thesis we adhere to a series of notational conventions, which are described briefly in this section.

The indicator function $\mathbb{1}\{P\}$ is defined as

$$\mathbb{1}\{P\} = \begin{cases} 1 & \text{if } P \text{ is true} \\ 0 & \text{otherwise,} \end{cases} \tag{2.1}$$

for any boolean expression P. The powerset of a set X is denoted by $\mathfrak{P}(X)$.

Generally, vectors are represented by lowercase boldface letters, matrices by uppercase boldface letters. Their respective elements are written using their non-boldface counterparts with subscript indices. To give an example: Let $\boldsymbol{A}$ denote an arbitrary matrix of size $n \times m$, and $\boldsymbol{a}$ a vector of size n, then a_i is the i-th element of $\boldsymbol{a}$ and A_{ij} the element of $\boldsymbol{A}$ in row i and column j. A special matrix is $\boldsymbol{I}_n$, the $n \times n$ identity matrix. The subscript may be omitted if the size is clear from context.

We allow for more advanced indexing through subvectors:

Definition 2.1 (Subvectors). *Let $n \in \mathbb{N}$, $I \subseteq \{1, \dots, n\}$ an ordered set with $|I| = m$ and elements $I_1, \dots, I_m$, and $\boldsymbol{a} \in K^n$ for some field K. Then $\boldsymbol{a}_I$ is a* subvector *of $\boldsymbol{a}$ defined as*

$$\boldsymbol{a}_I = (a_{I_1}, \dots, a_{I_m})^\top .$$

By default, all vectors are column vectors, and superscript $\top$ denotes transposition for both vectors and matrices, e.g., $\boldsymbol{a}^\top$ and $\boldsymbol{A}^\top$. If not specified otherwise, simple addition and subtraction of (equally-sized) vectors and matrices is element-wise. The notation $\odot$ denotes the Hadamard product, i.e., element-wise multiplication. Addition and subtraction of scalars with vectors and matrices is "broadcast" to the size of the latter: The notation $\boldsymbol{A}+c$ (or $c+\boldsymbol{A}$) means that the scalar c is added to every element of $\boldsymbol{A}$, yielding a matrix of the same shape as $\boldsymbol{A}$.

Another operation that we use frequently throughout this thesis is the *Kronecker product*, which is commonly found in the context of quantum computing.

Definition 2.2 (Kronecker product)**.** *Given matrices $\boldsymbol{A} \in K^{n\times m}$ and $\boldsymbol{B} \in K^{u\times v}$ over some field K, their* Kronecker product *$\boldsymbol{A} \otimes \boldsymbol{B} = \boldsymbol{C}$ is a matrix in $K^{nu\times mv}$ with elements $C_{in+k-n,jm+\ell-m} = A_{ij}B_{k\ell}$ for $i \in \{1,\dots,n\}$, $j \in \{1,\dots,m\}$, $k \in \{1,\dots,u\}$ and $\ell \in \{1,\dots,v\}$. Intuitively, a copy of $\boldsymbol{B}$ is multiplied to each scalar element of $\boldsymbol{A}$, as shown in the following example:*

$$\boldsymbol{A} = \begin{bmatrix} 4 & -6 \\ 0 & -9 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} -10 & 8 & 3 \\ 8 & 5 & -5 \end{bmatrix}$$

$$\begin{aligned}
\boldsymbol{A} \otimes \boldsymbol{B} &= \begin{bmatrix} 4\boldsymbol{B} & -6\boldsymbol{B} \\ 0\boldsymbol{B} & -9\boldsymbol{B} \end{bmatrix} \\
&= \begin{bmatrix} 4\begin{bmatrix} -10 & 8 & 3 \\ 8 & 5 & -5 \end{bmatrix} & -6\begin{bmatrix} -10 & 8 & 3 \\ 8 & 5 & -5 \end{bmatrix} \\ 0\begin{bmatrix} -10 & 8 & 3 \\ 8 & 5 & -5 \end{bmatrix} & -9\begin{bmatrix} -10 & 8 & 3 \\ 8 & 5 & -5 \end{bmatrix} \end{bmatrix} \\
&= \begin{bmatrix} -40 & 32 & 12 & 60 & -48 & -18 \\ 32 & 20 & -20 & -48 & -30 & 30 \\ 0 & 0 & 0 & 90 & -72 & -27 \\ 0 & 0 & 0 & -72 & -45 & 45 \end{bmatrix}.
\end{aligned}$$

In contrast to the usual matrix multiplication, the Kronecker product can be applied to any pair of matrices, regardless of their sizes. The notation $\boldsymbol{A}^{\otimes k}$ is shorthand for $\boldsymbol{A}\otimes\boldsymbol{A}\otimes\cdots\otimes\boldsymbol{A}$ repeated k times. Note that the size of $\boldsymbol{A}^{\otimes k}$ grows exponentially with k.

The function diag[·] has two definitions, depending on its input: If $\boldsymbol{A}$ is a square matrix, diag[$\boldsymbol{A}$] gives the main diagonal as a vector, i.e., $\text{diag}[\boldsymbol{A}] = (A_{1,1}, A_{2,2}, \dots, A_{n,n})^\intercal$. On the other hand, given a vector $\boldsymbol{a}$ of length n, diag[$\boldsymbol{a}$] constructs a square diagonal matrix with $\boldsymbol{a}$ along its main diagonal, and with 0 in every off-diagonal position, i.e., $\text{diag}[\boldsymbol{a}] = \boldsymbol{M}$ with $M_{ij} = a_i$ if $i = j$, and $M_{ij} = 0$ otherwise, for all $i, j \in \{1,\dots,n\}$.

In the context of quantum computing, we often encounter complex numbers: The set of complex numbers is $\mathbb{C}$, and their elements are represented equivalently as either $a + b\text{i}$ (cartesian form) or $r \cdot e^{\text{i}\varphi} = r(\cos\varphi + \text{i}\sin\varphi)$ (polar form), with $a, b, r \in \mathbb{R}$, $\varphi \in [0, 2\pi)$, and i as the imaginary unit characterized by $\text{i}^2 = -1$. The real part of a complex number is defined as $\text{Re}(a + b\text{i}) = a$, and its imaginary part as $\text{Im}(a + b\text{i}) = b$.

The conjugate transpose of complex matrices is denoted by $\boldsymbol{A}^\dagger$, which is an $m\times n$ matrix with elements $A^\dagger_{ji} = A^*_{ij}$ for all $i \in \{1,\dots,n\}$ and $j \in \{1,\dots,m\}$. The notation c^* for any $c \in \mathbb{C}$ with $c = a + b\mathrm{i}$ denotes the complex conjugate defined as $(a + b\mathrm{i})^* = a - b\mathrm{i}$. In polar form, this becomes $(r \cdot e^{\mathrm{i}\varphi})^* = r \cdot e^{-\mathrm{i}\varphi}$.

2.1.1. Binary Vectors

Binary vectors (or *bit vectors*) are a type of vectors used frequently throughout this thesis. The set of binary vectors of length n are denoted by $\mathbb{B}^n$, where $\mathbb{B} = \{0, 1\}$ is the set of binary digits, also called *bits*. Instead of writing, e.g., $(0, 1, 0, 0, 1)^\intercal$, the more concise notation `01001` is used for constant bit vectors.

To allow for easier definitions later on, let us quickly recall a useful connection between binary vectors and index sets.

Definition 2.3. *Let $\iota_n : \mathbb{B}^n \to \mathfrak{P}\{1, \dots, n\}$ with*

$$\iota_n(\boldsymbol{z}) = \{i \in \{1, \dots, n\} : \ z_i = 1\},$$

which for a given bit vector $\boldsymbol{z}$ returns the set of indices i where $z_i = 1$. The sets $\mathbb{B}^n$ and $\mathfrak{P}\{1, \dots, n\}$ are isomorphic, their isomorphism being ι_n with its inverse

$$\iota_n^{-1}(I) = (\mathbb{1}\{1 \in I\}, \mathbb{1}\{2 \in I\}, \dots, \mathbb{1}\{n \in I\})^\intercal.$$

Identifying bit vectors with index sets is often convenient. Special bit vectors are the zero vector $\mathbf{0}_n = \iota_n^{-1}(\emptyset)$, the one vector $\mathbf{1}_n = \iota_n^{-1}(\{1, \dots, n\})$, and the unit vectors $\boldsymbol{e}_i^n = \iota_n^{-1}(\{i\}) \, \forall i \in \{1, \dots, n\}$. The superscript vector length is omitted when it is clear from context.

The norm $\|\boldsymbol{z}\|_1 \in \{0, \dots, n\}$ of a bit vector $\boldsymbol{z} \in \mathbb{B}^n$ is equal to its number of 1-bits, which is also known as its *Hamming weight*. The *Hamming distance* between two vectors $\boldsymbol{z}, \boldsymbol{y} \in \mathbb{B}^n$ is the number of indices i where $z_i \neq y_i$, denoted by $d_H(\boldsymbol{z}, \boldsymbol{y}) = \|\boldsymbol{z}\|_1 + \|\boldsymbol{y}\|_1 - 2\|\boldsymbol{z} \odot \boldsymbol{y}\|_1 = \sum_{i=1}^n z_i + y_i - 2z_i y_i$. Equivalently, it is the number of bits that need to be flipped to turn $\boldsymbol{z}$ into $\boldsymbol{y}$.

For later use in Chapter 6, we introduce notation for fixing a certain subvector of all binary vectors to a specific bit pattern:

Definition 2.4 (Binary vector subspaces)**.** *Let $n \in \mathbb{N}$, $I \subseteq \{1, \dots, n\}$ an ordered set with $|I| = m$, and $\boldsymbol{b} \in \mathbb{B}^m$. We define*

$$\mathbb{B}^n_{I \leftarrow \boldsymbol{b}} = \{\boldsymbol{z} \in \mathbb{B}^n \,| \boldsymbol{z}_I = \boldsymbol{b}\}.$$

2.2. Optimization

At the core of many, if not most, problems in computer science – and Machine Learning in particular – lie optimization problems. In a very general form they consist of some domain

$\mathcal{X}$ of solution candidates and a function $\mathcal{L} : \mathcal{X} \to \mathbb{R}$ called *loss function*, that assigns a value to each solution candidate indicating how “bad” it is [29]. In other contexts, such as Reinforcement Learning, we often find evaluating functions that measure how “good” a solution is; these functions are more commonly called *fitness functions*, and they are *maximized* during the optimization process. Now, the task is to find an $x \in \mathcal{X}$ such that $\mathcal{L}(x) = y$ is either *(i)* globally minimal, i.e., $y = \min_{x' \in \mathcal{X}} \mathcal{L}(x')$, or *(ii)* locally minimal, i.e., $y = \min_{x'' \in \mathcal{N}(x)} \mathcal{L}(x'')$, where $\mathcal{N}(x) \subseteq \mathcal{X}$ describes some neighborhood or vicinity around x [29, Sec. 1.6]. For instance, if $\mathcal{X} = \mathbb{B}^n$ for some $n > 0$, a neighborhood $\mathcal{N}(\boldsymbol{z}; k)$ might be $\{\boldsymbol{z}' \in \mathbb{B}^n : d_H(\boldsymbol{z}, \boldsymbol{z}') \leq k\}$ for some fixed $k \in \{1, \ldots, n\}$, i.e., all bit vectors differing in at most k bits to the original vector. If $\mathcal{X} = \mathbb{R}^d$, a neighborhood might be $\mathcal{N}(\boldsymbol{r}; \epsilon) = \{\boldsymbol{r}' \in \mathbb{R}^d : \|\boldsymbol{r} - \boldsymbol{r}'\|_2 \leq \epsilon\}$, i.e., an ϵ-hypersphere around any given point.

As an extension, some optimization problems employ constraints on the solution space, forbidding certain solutions $x \in \mathcal{C} \subset \mathcal{X}$. In the above framework, we can incorporate such constraints into $\mathcal{L}$ by defining

$$\mathcal{L}_{\mathcal{C}}(x; \lambda) = \begin{cases} \lambda & \text{if } x \in \mathcal{C}, \\ \mathcal{L}(x) & \text{otherwise.} \end{cases}$$

Here, $\lambda > 0$ is a value that needs to be chosen large enough to render any solution candidate $x \in \mathcal{C}$ unfit as a local or global optimum, e.g., by choosing $\lambda = \sup \mathcal{L}$ [29, Sec. 10.7]. However, computing or estimating this value can be tricky in practice.

Naturally, finding a globally minimizing solution is, in general, harder than finding a local one, particularly if $\mathcal{X}$ is large or infinite, and it is harder to give guarantees of global optimality [29].

2.2.1. Gradients in Optimization

If the loss function is differentiable, we can use its gradient to perform optimization [29, Sec. 5.1]. The gradient of a function $g : \mathbb{R}^d \to \mathbb{R}$ at a point $\boldsymbol{r} \in \mathbb{R}^d$ is defined as

$$\nabla g(\boldsymbol{r}) = \left(\frac{\partial g}{\partial r_1}(\boldsymbol{r}), \ldots, \frac{\partial g}{\partial r_d}(\boldsymbol{r}) \right)^{\mathsf{T}}, \tag{2.2}$$

which is the vector of all partial derivatives w.r.t. every element of the input vector. Inuitively, this vector in d-dimensional space points to the direction of steepest ascent [9, Sec. 10.10.1].

Assume that $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$. As we wish to minimize the loss function $\mathcal{L}$, we can start from some initial position $\boldsymbol{r}^0$, compute the gradient $\nabla \mathcal{L}(\boldsymbol{r})$ and take a step in the *opposite* direction with a certain step size $\eta > 0$,

$$\boldsymbol{r}^{t+1} = \boldsymbol{r}^t - \eta \nabla \mathcal{L}(\boldsymbol{r}^t), \tag{2.3}$$

for $t \geq 0$. In the context of Machine Learning, η is often referred to as the *learning rate* [9, 29]. If η is chosen small and T large enough, we can expect that $\boldsymbol{r}^1, \boldsymbol{r}^2, \ldots, \boldsymbol{r}^T$ reaches a low point of $\mathcal{L}$.

This method is known as Gradient Descent (GD), and is applicable whenever the candidate space $\mathcal{X}$ is real-valued and the loss function is differentiable. A sensible starting point should not be too far away from an optimum, so that it can be reached in a reasonable number of steps (also depending on the step size η). Finding such an initial point is not trivial.

2.2.2. Evolutionary Optimization

EAs are a family of optimization algorithms inspired by the principles of natural selection and genetics. At the core, EAs involve a population of candidate solutions, which evolve over successive iterations (often called *generations*) to improve on an objective function. While many classical optimization methods rely on gradients or convexity assumptions, EAs are inherently heuristic, making them well-suited for tackling complex, non-linear, and multi-modal optimization problems that are common in ML and other domains. They have been researched intensively and applied to a wide range of optimization scenarios, including programming and formal grammars (see [30] for a detailed overview).

Algorithm 2.1 Generic EA after Bäck and Schwefel [30]; the additional individuals $\tilde{P}$ passed to the selection operator is either $\emptyset$, leading to comma selection, or $P(t)$, leading to plus selection.

```
$t \leftarrow 0$
Initialize population: $P(t) = \{\boldsymbol{a}_1(t), \ldots, \boldsymbol{a}_\mu(t)\}$
Evaluate population: $\{f(\boldsymbol{a}_1(t)), \ldots, f(\boldsymbol{a}_\mu(t))\}$
while termination criterion not reached do
    Recombine: $P'(t) = \mathfrak{r}(P(t)) = \{\boldsymbol{a}'_1(t), \ldots, \boldsymbol{a}'_\lambda(t)\}$
    Mutate: $P''(t) = \mathfrak{m}(P'(t)) = \{\boldsymbol{a}''_1(t), \ldots, \boldsymbol{a}''_\lambda(t)\}$
    Evaluate offspring: $\{f(\boldsymbol{a}''_1(t)), \ldots, f(\boldsymbol{a}''_\lambda(t))\}$
    Select: $P(t+1) = \mathfrak{s}(P(t) \cup \tilde{P})$
    $t \leftarrow t + 1$
end while
```

A generic EA is sketched in Algorithm 2.1, adapted from [30]: The algorithm begins with a population of $\mu > 0$ candidate solutions, which are typically randomly initialized. Each solution is evaluated against a fitness function f that measures its quality or performance. The evolution of solutions is driven by two main genetic operators:

- **Recombination** (or Crossover) simulates sexual reproduction, where two or more parent solutions combine to create $\lambda \geq \mu$ offspring. Recombination introduces diversity by mixing the characteristics of the parents, allowing the offspring to potentially explore new regions of the solution space. For instance, in a binary representation, crossover might involve swapping segments of two parent vectors to generate new candidate solutions.

- **Mutation** introduces random, small changes to individual candidates. For example, flipping a bit in a binary vector or perturbing a continuous variable. The role of mutation is to maintain genetic diversity within the population, preventing premature convergence to local optima. While recombination tends to explore combinations of existing good traits, mutation ensures that the search process retains the ability to explore unexplored areas of the solution space.

The population evolves by iteratively selecting the fittest individuals, allowing them to survive and reproduce, mimicking Darwinian survival of the fittest. This selection process creates a pressure toward finding better solutions over time. In $(\mu + \lambda)$ and (μ, λ) evolutionary strategies, we find two common approaches to managing the population between generations:

- For **plus selection** (or $(\mu + \lambda)$ selection), the new population is formed by selecting the best solutions from both the current population (parents) and the newly generated offspring. This means that the previous generation can still contribute to the next, allowing high-quality solutions to persist indefinitely. This strategy ensures stability and often leads to faster convergence, as the best individuals are retained. The property of an EA that the best-performing solutions always survive to the next generation is called *elitism*. It accelerates convergence, but carries the risk that diversity decreases too quickly, especially in small populations, potentially leading to premature convergence to suboptimal solutions. This type of EA is among the most representative and best-understood in literature [31, 32].

- For **comma selection** (or (μ, λ) selection), the next generation is formed exclusively from the offspring, with no survivors from the parent population. This increases the algorithm's exploratory behavior, as the population is constantly refreshed. However, this method can lead to loss of valuable solutions if the offspring do not outperform their predecessors, making it a more aggressive exploration mechanism. Here, *forced elitism* can be used, where the globally best-performing solution ever encountered is always saved separately from the population.

2.2.3. Quadratic Unconstrained Binary Optimization

A specific optimization problem that is in the focus of this thesis is *quadratic unconstrained binary optimization* (QUBO). It can be considered the border stone between classical and quantum optimization, as *(i)* a multitude of classical optimization problems can be reduced to it, *(ii)* its solution can be approximated by classical optimization schemes and heuristics, and *(iii)* it can be solved using Quantum Annealing on quantum hardware. The last point is due to its close connection to the *Ising model* used in statistical physics, which will be elaborated on later in this section.

Its great value lies in its applicability to a wide range of combinatorial optimization problems, from economics [33, 34] over satisfiability [35], resource allocation and routing problems [36, 37] to machine learning [38, 7, 39, 5] – just to name a few. Its structural simplicity has made it a popular target problem for special-purpose hardware solvers [40, 41].

Definition 2.5 (QUBO [42]). *Let $n \in \mathbb{N}$ and $\boldsymbol{Q} \in \mathcal{Q}_n$, where $\mathcal{Q}_n \subseteq \mathbb{R}^{n\times n}$ denotes the set of all upper-triangular real-valued square matrices of size $n \times n$. We define the* energy *of a binary vector $\boldsymbol{z} \in \mathbb{B}^n$ w.r.t. $\boldsymbol{Q}$ as*

$$f_{\boldsymbol{Q}}(\boldsymbol{z}) = \boldsymbol{z}^\intercal \boldsymbol{Q} \boldsymbol{z} = \sum_{i=1}^{n}\sum_{j=1}^{n} Q_{ij} z_i z_j .$$

Now, QUBO is the problem of finding a vector $\boldsymbol{z}^* \in \mathbb{B}^n$ that has minimal energy w.r.t. $f_{\boldsymbol{Q}}$, i.e., $\forall \boldsymbol{z} \in \mathbb{B}^n : \; f_{\boldsymbol{Q}}(\boldsymbol{z}^*) \leq f_{\boldsymbol{Q}}(\boldsymbol{z})$.

A QUBO instance is fully specified by the matrix $\boldsymbol{Q}$, which is called *weight matrix* or *parameter matrix* [42]. $\mathcal{Q}_n$ is the set of all valid QUBO weight matrices, and therefore the set of all possible QUBO instances of size n. In the course of this thesis, the terms "QUBO instance" and "QUBO weight matrix" may be used interchangeably at times for this reason, though they are technically distinct mathematical objects.

In literature we frequently encounter alternative definitions of QUBO, all of which are equivalent to the one given above. The following list gives an overview of alternative definitions g and their equivalent formulations using f, which are derived through simple rearrangement:

- $g_{\boldsymbol{Q}}(\boldsymbol{z}) = \boldsymbol{z}^\intercal \boldsymbol{Q} \boldsymbol{z}$, where $g_{\boldsymbol{Q}}$ is to be *maximized* instead of minimized. Flipping the sign of all elements of $\boldsymbol{Q}$ yields the equivalent minimization problem $f_{-\boldsymbol{Q}}$.
- $g_{\boldsymbol{Q},\boldsymbol{c}}(\boldsymbol{z}) = \boldsymbol{z}^\intercal \boldsymbol{Q} \boldsymbol{z} + \boldsymbol{z}^\intercal \boldsymbol{c}$, with separate linear and quadratic terms, which is equivalent to $f_{\boldsymbol{Q}+\mathrm{diag}[\boldsymbol{c}]}$.
- $g_{\boldsymbol{R}}(\boldsymbol{z}) = \boldsymbol{z}^\intercal \boldsymbol{R} \boldsymbol{z}$, where $\boldsymbol{R} \in \mathbb{R}^{n\times n}$ is a non-triangular matrix. The definition of $f_{\boldsymbol{R}}$ in Def. 2.5 still works for such matrices, and our assumption of upper-triangular weight matrices is simply a convention.

In addition, above definitions may accur in every combination, i.e., separate linear and quadratic terms with a non-triangular matrix. Having explicit linear and quadratic coefficients may, at times, be a bit more intuitive and help clarity, whereas our definition needs the special notation diag$[\boldsymbol{Q}]$ to refer to the linear terms. However, the disadvantage of many alternative formulations is that they do not uniquely define the energy function, whereas for Def. 2.5 there is a one-to-one correspondence between $\boldsymbol{Q}$ and $f_{\boldsymbol{Q}}$ for all $\boldsymbol{Q} \in \mathcal{Q}_n$.

A QUBO instance may have multiple minimizing vectors, which leads to the following definition of optimal sets:

Definition 2.6 (Optimal set [1]). *Let $\boldsymbol{Q} \in \mathcal{Q}_n$. The set of minimizing vectors (or* optimal set*) w.r.t. to $\boldsymbol{Q}$ is defined as*

$$S^*(\boldsymbol{Q}) = \{\boldsymbol{z} \in \mathbb{B}^n \mid f_{\boldsymbol{Q}}(\boldsymbol{z}) \leq f_{\boldsymbol{Q}}(\boldsymbol{z}') \, \forall \boldsymbol{z}' \in \mathbb{B}^n\}.$$

Equivalently, $S^*(\boldsymbol{Q}) = \{\boldsymbol{z} \in \mathbb{B}^n \mid f_{\boldsymbol{Q}}(\boldsymbol{z}) = f^*(\boldsymbol{Q})\}$, where $f^*(\boldsymbol{Q}) = \min_{\boldsymbol{z} \in \mathbb{B}^n} f_{\boldsymbol{Q}}(\boldsymbol{z})$ is the globally minimal energy.

Using this definition, we can define another helpful notion that will be used extensively in Chapter 6:

Definition 2.7 (Optimum inclusion [1]). *Let $\boldsymbol{Q}, \boldsymbol{R} \in \mathcal{Q}_n$ for any $n \in \mathbb{N}$. We say that $\boldsymbol{Q}$* includes the optima of $\boldsymbol{R}$, *written as $\boldsymbol{Q} \sqsubseteq \boldsymbol{R}$, if $S^*(\boldsymbol{Q}) \subseteq S^*(\boldsymbol{R})$. If both $\boldsymbol{Q} \sqsubseteq \boldsymbol{R}$ and $\boldsymbol{R} \sqsubseteq \boldsymbol{Q}$, we say that $\boldsymbol{Q}$ and $\boldsymbol{R}$ are* optimum-equivalent, *written as $\boldsymbol{Q} \equiv \boldsymbol{R}$.*

As a simple example, it is clear that $\boldsymbol{Q} \equiv \alpha \boldsymbol{Q} \; \forall \alpha > 0$, as the minimum function is invariant w.r.t. scaling with a positive factor. In general, however, it is just as hard to determine if $\boldsymbol{Q} \sqsubseteq \boldsymbol{R}$ for any given $\boldsymbol{Q}, \boldsymbol{R} \in \mathcal{Q}_n$ as solving the Qubo problem itself, namely NP-hard.

Proposition 2.1. Determining $\boldsymbol{Q} \sqsubseteq \boldsymbol{R}$ for any $\boldsymbol{Q}, \boldsymbol{R} \in \mathcal{Q}_n$ is NP-hard, if $\boldsymbol{Q}$ has a unique minimizer.

Proof. Our proof follows from a polynomial-time reduction to Qubo, which is known to be NP-hard (see paragraph about hardness below). Let $\boldsymbol{E}_i = -\boldsymbol{e}_i \boldsymbol{e}_i^\intercal \in \mathcal{Q}_n$ for all $i \in \{1, \dots, n\}$; clearly, $S^*(\boldsymbol{E}_i) = \{\boldsymbol{z} \in \mathbb{B}^n \mid z_i = 1\}$. Furthermore, we observe that for any $\boldsymbol{b} \in \mathbb{B}^n$ we have $\bigcap_{i=1}^n \{\boldsymbol{z} \in \mathbb{B}^n \mid z_i = b_i\} = \{\boldsymbol{b}\}$. Now, let $\boldsymbol{z}^* = (\mathbb{1}\{\boldsymbol{Q} \sqsubseteq \boldsymbol{E}_1\}, \dots, \mathbb{1}\{\boldsymbol{Q} \sqsubseteq \boldsymbol{E}_n\})^\intercal$, whose entries can be computed efficiently according to our assumption. It is easy to see that $\boldsymbol{z}^*$ is the minimizer of $\boldsymbol{Q}$, as we were able to "probe" every bit of the solution vector. As we computed $\boldsymbol{z}^*$ in n steps, we reduced Qubo to the decision problem $\boldsymbol{Q} \sqsubseteq \boldsymbol{R}$ in polynomial time, from which NP-hardness of the latter follows. $\square$

It is not very restrictive to assume that a given Qubo instance has only one unique minimizing vector. This can be justfied by the intuition that a small perturbation of the weight matrix can cause the the energy values in $S^*(\boldsymbol{Q})$ to "tip out of balance".

Proposition 2.2. For every $\boldsymbol{Q} \in \mathcal{Q}_n$ and $\epsilon > 0$, there is always a weight matrix $\tilde{\boldsymbol{Q}} \sqsubseteq \boldsymbol{Q}$ with $\|\boldsymbol{Q} - \tilde{\boldsymbol{Q}}\| \leq \epsilon$ and $|S^*(\tilde{\boldsymbol{Q}})| = 1$, where $\|\cdot\|$ is any matrix norm.

Proof. Given $\boldsymbol{Q} \in \mathcal{Q}_n$ and $\epsilon > 0$, choose $\boldsymbol{z}^* = \arg\max_{\boldsymbol{z} \in S^*(\boldsymbol{Q})} \|\boldsymbol{z}\|_1$. Now, define $\boldsymbol{\Delta} = \operatorname{diag}[-\epsilon \cdot \boldsymbol{z}^*]$. By construction, $f_{\boldsymbol{\Delta}}(\boldsymbol{z}) = -\epsilon \cdot \|\boldsymbol{z}\|_1$ is minimal only for $\boldsymbol{z}^*$ within $S^*(\boldsymbol{Q})$. Set $\tilde{\boldsymbol{Q}} = \boldsymbol{Q} + \boldsymbol{\Delta}$, and the proof follows from the optimality of $\boldsymbol{z}^*$ w.r.t. $\boldsymbol{Q}$ and the equivalence of all matrix norms. $\square$

Hardness of Qubo In its general form, Qubo is known to be strongly NP-hard [43, 44], and its exact solution requires, in the worst case, an exhaustive search of an exponentially large candidate space. A range of solution techniques has been developed over past decades. Some obtain the exact result [45, 42, 46] with worst-case exponential running time. Faster approximate techniques range from linear constraint solvers to heuristics such as simulated annealing [47], tabu search [48], and genetic programming [49]. A very comprehensive overview of solution techniques, both exact and approximative, can be found in [50].

Perhaps most remarkably, Qubo can be mapped to an Ising model [51] and solved through quantum annealing, which exploits quantum tunneling effects [17]. This close connection to the Ising model shall be highlighted in more detail.

Connection to the Ising Model A computational model that is structurally very similar to Qubo is the *Ising model*, named after physicist Ernst Ising, sometimes also called "Lenz-Ising model" after its inventor, Wilhelm Lenz [52, 51]. Its original purpose is to model magnetic spins of atomic particles arranged in a lattice, but, just like Qubo, it can be applied much more generally.

Definition 2.8 (Ising model [42]). *Let $n \in \mathbb{N}$, $\boldsymbol{J} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{h} \in \mathbb{R}^n$. The energy of a binary vector $\boldsymbol{\sigma} \in \mathbb{S}^n$ with $\mathbb{S} = \{-1, +1\}$ w.r.t. $\boldsymbol{J}$ and $\boldsymbol{h}$ is given by the* Hamiltonian function

$$H_{\boldsymbol{J},\boldsymbol{h}}(\boldsymbol{\sigma}) = \boldsymbol{\sigma}^\intercal \boldsymbol{J}\boldsymbol{\sigma} + \boldsymbol{h}^\intercal\boldsymbol{\sigma} = \sum_{i=1}^{n}\sum_{j=1}^{n} J_{ij}\sigma_i\sigma_j + \sum_{i=1}^{n} h_i\sigma_i.$$

The probability of a configuration $\boldsymbol{\sigma}$ is given by the Boltzmann distribution with an inverse temperature $\beta \propto 1/T \geq 0$,

$$P_{\boldsymbol{J},\boldsymbol{h}}(\boldsymbol{\sigma};\beta) = e^{-\beta H_{\boldsymbol{J},\boldsymbol{h}}(\boldsymbol{\sigma})}/Z_{\boldsymbol{J},\boldsymbol{h}} \text{ with } Z_{\boldsymbol{J},\boldsymbol{h}}(\beta) = \sum_{\boldsymbol{\sigma}'\in\mathbb{S}^n} e^{-\beta H_{\boldsymbol{J},\boldsymbol{h}}(\boldsymbol{\sigma}')},$$

where $Z_{\boldsymbol{J},\boldsymbol{h}}(\beta)$ is the partition function *acting as the normalization constant.*

The weights J_{ij} are called *interactions*, and h_i the external (magnetic) field strengths. They are the equivalent of the quadratic and linear terms in the Qubo energy function in Def. 2.5. In the original model, the variables form a lattice, i.e., every σ_i has only up to four neighbors. Such structures can be realized through sparse interaction matrices.

Sometimes the Ising Hamiltonian function is defined with a negative sign, which stems from a physical convention in its application to ferromagnetic systems. Additionally, a scalar magnetic moment μ is sometimes multiplied to $\boldsymbol{h}^\intercal\boldsymbol{\sigma}$, which balances the interactions and external field strengths. Both can be incorporated into $\boldsymbol{J}$ and $\boldsymbol{h}$ without loss of generality, which is why we assume the form given above throughout this thesis.

It is easy to see that there is a simple linear mapping between Qubo and the Ising model, which shows that the two are equivalent.

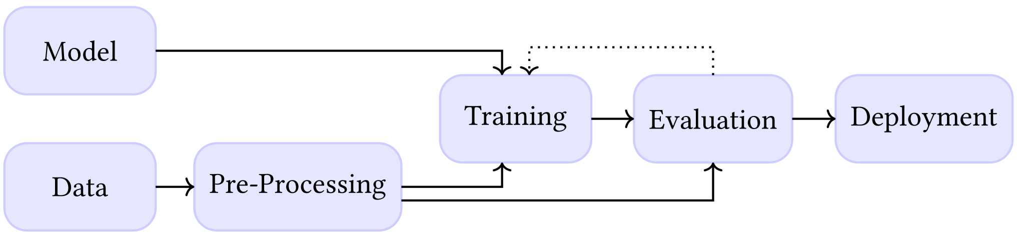


Figure 2.1.: Generalized machine learning pipeline.

Proposition 2.3. Given any Ising model with interactions $\boldsymbol{J} \in \mathbb{R}^{n\times n}$ and external field strengths $\boldsymbol{h} \in \mathbb{R}^n$, there is a QUBO weight matrix $\boldsymbol{Q}$ and a constant offset $c \in \mathbb{R}$, such that

$$\forall\, \boldsymbol{z} \in \mathbb{B}^n :\ H_{\boldsymbol{J},\boldsymbol{h}}(\mathbf{1} - 2\boldsymbol{z}) = f_{\boldsymbol{Q}}(\boldsymbol{z}) + c, \tag{2.4}$$

where $f_{\boldsymbol{Q}}$ is defined as in Def. 2.5. Conversely, for any given $\boldsymbol{Q} \in \mathcal{Q}_n$, we can find $\boldsymbol{J}$, $\boldsymbol{h}$ and c such that Eq. (2.4) holds as well.

Proof. Let $n \in \mathbb{N}$, $\boldsymbol{J} \in \mathbb{R}^{n\times n}$ and $\boldsymbol{h} \in \mathbb{R}^n$ be arbitrary but fixed. Note that the mapping $\boldsymbol{z} \mapsto \mathbf{1} - 2\boldsymbol{z}$ is an isomorphism between $\mathbb{B}^n$ and $\mathbb{S}^n$, mapping entries 1 to -1 and 0 to $+1$. Through simple rearrangement we obtain

$$\begin{aligned}
H_{\boldsymbol{J},\boldsymbol{h}}(\mathbf{1} - 2\boldsymbol{z}) &= (\mathbf{1} - 2\boldsymbol{z})^\intercal \boldsymbol{J}(\mathbf{1} - 2\boldsymbol{z}) + \boldsymbol{h}^\intercal(\mathbf{1} - 2\boldsymbol{z}) \\
&= 4\boldsymbol{z}^\intercal\boldsymbol{J}\boldsymbol{z} - 2\boldsymbol{z}^\intercal\boldsymbol{J}\mathbf{1} - 2\mathbf{1}^\intercal\boldsymbol{J}\boldsymbol{z} + \mathbf{1}^\intercal\boldsymbol{J}\mathbf{1} - 2\boldsymbol{h}^\intercal\boldsymbol{z} + \boldsymbol{h}^\intercal\mathbf{1} \\
&= \boldsymbol{z}^\intercal(4\boldsymbol{J})\boldsymbol{z} - 2(\boldsymbol{J}\mathbf{1} + \boldsymbol{J}^\intercal\mathbf{1} + \boldsymbol{h})^\intercal\boldsymbol{z} + \underbrace{\boldsymbol{h}^\intercal\mathbf{1} + \mathbf{1}^\intercal\boldsymbol{J}\mathbf{1}}_{=:c} \\
&= \boldsymbol{z}^\intercal\underbrace{\left(4\boldsymbol{J} - 2\operatorname{diag}[\boldsymbol{J}\mathbf{1} + \boldsymbol{J}^\intercal\mathbf{1} + \boldsymbol{h}]\right)}_{=:\boldsymbol{Q}}\boldsymbol{z} + c \\
&= f_{\boldsymbol{Q}}(\boldsymbol{z}) + c.
\end{aligned}$$

To ensure that $\boldsymbol{Q}$ is upper-triangular, simply map $Q_{ij} \mapsto Q_{ij} + Q_{ji}$ and $Q_{ji} \mapsto 0$ for all $i < j$. The opposite direction, i.e., obtaining $\boldsymbol{J}$ and $\boldsymbol{h}$ from a given $\boldsymbol{Q}$ is analogous by applying $\boldsymbol{z} \mapsto (\mathbf{1} - \boldsymbol{\sigma})/2$, which yields

$$f_{\boldsymbol{Q}}((\mathbf{1} - \boldsymbol{\sigma})/2) = \boldsymbol{\sigma}^\intercal\underbrace{(\boldsymbol{Q}/4)}_{=:\boldsymbol{J}}\boldsymbol{\sigma} + \underbrace{(-(\boldsymbol{Q}\mathbf{1} + \boldsymbol{Q}^\intercal\mathbf{1})/4)}_{=:\boldsymbol{h}}{}^\intercal\boldsymbol{\sigma} + \underbrace{\mathbf{1}^\intercal\boldsymbol{Q}\mathbf{1}/4}_{=:-c}.$$

□

2.3. Machine Learning

For many complex tasks, we cannot easily find precise mathematical models or develop algorithms that solve them exactly. A number of such tasks require, by human standards, a certain degree of intelligence, like recognizing shapes and objects in images, finding patterns in data, or making predictions based on prior observations. Over the past 80 years, numerous methods have been developed to tackle such tasks nonetheless, opening the research area of ML. Over the past few decades, due to leaps in hardware development, ML has gained greater relevance than ever and is a key technology in a wide range of applications.

At its core, ML uses optimization techniques with the goal of learning from data by *(i)* collecting a *data set* $\mathcal{D}$ containing examples, *(ii)* defining a *model* $f_{\boldsymbol{\theta}}$ with parameters $\boldsymbol{\theta}$, *(iii)* defining a loss function on the model's output, and *(iv)* iteratively improving the model's loss by updating $\boldsymbol{\theta}$ based on the examples in $\mathcal{D}$.

If the data set consists of pairs of input data points and target values, and the model is supposed to predict the target values from any given input, we arrive at the definition of *supervised learning* [9, Sec. 2].

Definition 2.9 (Supervised Learning [9]). *Let $\mathbb{D}$ be a joint distribution over $\mathcal{X} \times \mathcal{Y}$, and $\mathcal{D} \subset \mathcal{X} \times \mathcal{Y}$ a sample drawn from $\mathbb{D}$ with $|\mathcal{D}| = N$, containing pairs $(\boldsymbol{x}^i, y_i)$ of input data points $\boldsymbol{x}^i \in \mathcal{X}$ and* labels *$y_i \in \mathcal{Y}$ for $i \in \{1, \ldots, N\}$. The data points can be interpreted as rows of a matrix $\boldsymbol{X}$ such that $\boldsymbol{X}_{i,\cdot} = \boldsymbol{x}^i$. The columns of this matrix are called* features. *The sample $\mathcal{D}$ is called a* data set. *Given $\mathcal{D}$, we want to find a function $f_{\boldsymbol{\theta}} : \mathcal{X} \to \mathcal{Y}$ with parameters $\boldsymbol{\theta}$ that takes data points and returns good predictions $\hat{y} = f(\boldsymbol{x})$ for any data point $\boldsymbol{x} \in \mathcal{X}$ (particularly outside the set $\mathcal{D}$), e.g.,*

$$f(\boldsymbol{x}) = \underset{y \in \mathcal{Y}}{\arg\max}\; p_{\mathbb{D}}(y \mid \boldsymbol{x}),$$

where $p_{\mathbb{D}}$ is the probability mass or density function w.r.t. distribution $\mathbb{D}$. If $\mathcal{Y}$ is continuous (e.g., $\mathcal{Y} = \mathbb{R}^d$), this problem is called regression. *Otherwise, if $\mathcal{Y}$ is discrete and finite (e.g., $\mathcal{Y} = \{1, \ldots, C\}$ for some $C \in \mathbb{N}$), the problem is called* classification.

A general ML pipeline is shown in Fig. 2.1 and consists of the following steps: A model is chosen from a family of models that is suitable for the classification or regression task at hand. The raw data is pre-processed to remove unnecessary features, discard or impute missing values, or augment existing features. The available data is then split into a training and test set, and the model's loss is minimized through an (iterative) optimization routine in a *training* phase, until the loss has converged or a fixed budget is depleted. Afterwards, the model is evaluated on the test or *evaluation* set to check if the model generalizes to newly observed data. If the performance is not satisfying, the model hyperparameters or pre-processing may be adapted and training repeated, until the performance meets the expectations, after which the model can be deployed.

Model Based on the task at hand, we need a parametric model that we can train. Popular models include ANNs, SVMs (see Section 2.3.2), Decision Trees (DTs) and Random Forests (RFs), but also simpler models such as Linear and Logistic Regression, as well as statistical models such as Gaussian Processes and Markov Random Fields [9]. If we have little or no prior knowledge about the data, the model function $f_{\boldsymbol{\theta}}$ should be able to approximate any function given the right set of parameters $\boldsymbol{\theta}$. The problem of minimizing the loss therefore reduces to finding appropriate parameters $\boldsymbol{\theta}$ w.r.t. to some data set:

$$\min_{\boldsymbol{\theta}} \mathcal{L}(f_{\boldsymbol{\theta}};\, \mathcal{D}_{\text{eval}}).$$

If the model has real-valued parameters, then this problem can be approached using gradient-based methods, such as gradient descent or stochastic gradient descent [29, Sec. 2]. Many models have both trainable parameters and *hyperparameters*: The former are adapted during the training procedure to minimize the loss function. The latter are fixed in advance depending on the input data, prior knowledge and performance requirements [29, Sec. 3.2]. Examples of hyperparameters include but are not limited to

- the number of input, output and hidden neurons in an ANN,
- the choice of activation function in an ANN,
- the misclassification penalty factor C in Support Vector Machines (see Section 2.3.2),
- the ansatz (gate structure) of a variational quantum circuit (see Section 2.4.2),
- the number k of cluster centers in k-means clustering.

Data As we have seen in Def. 2.9, we assume that the data stems from an underlying probability distribution $\mathbb{D}$. In practice, this true data distribution is unknown, but we can approximate it with the empirical distribution of the sample data set. For discrete $\mathcal{X}$ and $\mathcal{Y}$, the probability mass function can be simply found by counting,

$$\hat{p}_{\mathcal{D}}(\boldsymbol{x}, y) = \frac{1}{N} \sum_{(\boldsymbol{x}', y') \in \mathcal{D}} \mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x} \wedge y' = y\}. \tag{2.5}$$

The larger the data set, the more closely the distribution approaches the true data distribution. It is common practice to split a data set into two or more subsets, using one for training and one for evaluation in order to test the model's behavior on previously unobserved input values. If the model works well on the evaluation data set, we say that it *generalizes* well [9, Sec. 2.9]. On the other hand, if the model has a low loss values on the training data, but a very high loss on the evaluation data, we say that the model is *overfitting*.

Pre-Processing In most practical applications, the raw data cannot be immediately used for training, but has to be processed (see, e.g., [53]). This can be due to various reasons, such as

- vastly different value magnitudes that have to be normalized through **scaling**,
- an excessive amount of **noise**, which has to be mitigated, e.g., through low-pass filtering or smoothing using a moving average,

- feature columns containing **missing values**, which have to be imputed, or data points with missing values simply discarded; this is necessary in particular for gradient-based optimization, which cannot handle missing values when computing the gradient,
- the presence of **irrelevant features** with no or little predictive power, which can be discarded to save resources (see Section 2.3.1),
- the necessity of **feature engineering** by applying (non-linear) transformations to one or more features to create new features, often based on prior knowledge about the data or task,
- a low amount of data points requiring **data augmentation**, e.g., by modifying or transforming existing data points to obtain more training data; this is especially popular for image data that can be scaled, rotated, cropped or sheared.

After applying the necessary pre-processing steps, the data is ready to enter the training loop where the loss function is iteratively evaluated and minimized.

Loss Function As the data distribution $\mathbb{D}$ is generally unknown, a loss function $\mathcal{L}$ is used to quantify the quality of a given predictor. For instance, assume we have some data set $\mathcal{D}_{\text{train}} = \{(\boldsymbol{x}^i, y_i)\}_{i\in\{1,\dots,M\}}$, then a suitable loss function for a classification problem may be

$$\mathcal{L}_{\text{NegAcc}}(f; \mathcal{D}_{\text{train}}) = -\frac{1}{M}\sum_{i=1}^{M} \mathbb{1}\{f(\boldsymbol{x}^i) = y_i\}, \tag{2.6}$$

which is the negative classification accuracy. On the other hand, a loss function for a regression problem with $\mathcal{Y} = \mathbb{R}$ may be

$$\mathcal{L}_{\text{MSE}}(f; \mathcal{D}_{\text{train}}) = \frac{1}{M}\sum_{i=1}^{M}(f(\boldsymbol{x}^i) - y_i)^2, \tag{2.7}$$

commonly known as the Mean Squared Error (MSE). In addition to the primary objective of correct classification or regression, loss functions may contain *regularization* terms that penalize the model parameters, e.g., discouraging values of large magnitude by adding a p-norm of the parameter vector weighted with some positive factor λ,

$$\mathcal{L}_{\text{MSE,Reg}}(f_\Theta; \mathcal{D}_{\text{train}}) = \frac{1}{M}\sum_{i=1}^{M}(f(\boldsymbol{x}^i) - y_i)^2 + \lambda\|\Theta\|_2. \tag{2.8}$$

Different types of regularization lead to different model properties (see [9, Sec. 3.4]), most notably

1. **L1 regularization** (or "LASSO") encourages sparse parameter vectors, i.e., the larger λ is chosen, the more parameters will have a value of 0: $\|\boldsymbol{\theta}\|_1$

2. **L2 regularization** (or "ridge regression") encourage evenly distributed parameter values with small magnitudes by penalizing large outliers: $\|\boldsymbol{\theta}\|_2^2$

The choice of loss function and regularization has profound impact on the training process and the resulting model, its generalization properties and optimality guarantees. Making the right choice in practice is mostly an engineering task, as it depends on the interplay of the data and its properties, the model type, and the training procedure. Some regularization techniques are enacted directly on the model, independently from the loss function, such as drop-out in ANNs.

Training & Evaluation After choosing a model, setting its hyperparameters and preparing the training data, the training procedure is executed. For many models, this involves iteratively adapting the trainable parameters $\boldsymbol{\theta}$ of the model in order to minimize the loss function on a training set $\mathcal{D}_{\text{train}}$ [9]. An overwhelmingly popular approach is SGD, where the loss function is evaluated on a *batch* of training data (i.e., a subset of $\mathcal{D}_{\text{train}}$) and the parameters nudged in the opposite direction of the gradient with a learning rate $\eta > 0$ [29, Sec. 8.1]. The gradient computed from batches that are sampled uniformly from the training set is, in expectation, equal to the gradient computed on the entire training set, due to linearity of expectation. SGD reduces the computations load significantly, especially on very large data sets.

The training procedure is terminated if the loss value on the training data does no longer improve with each iteration, i.e., it has reached convergence, or if some fixed budget of function evaluations is reached. Particularly in deep neural network training, *early stopping* may be employed to prevent overfitting and achieve higher generalization [9, Sec. 11.5.2], i.e., a lower loss value on the evaluation data set. Instead of a single evaluation set, it is common to use *cross-evaluation*, which gives a more reliable estimate of the expected loss on unobserved data [9, Sec. 7.10]: In k-fold cross validation, a data set is split into k equally-sized partitions, and each partition in turn is set aside for evaluation while training is performed on the union of all remaining partitions. The resulting k evaluation loss values are averaged and gives a better impression of the model's loss on unseen data at the cost of repeating the training procedure k times.

A different ML scenario is *unsupervised learning*, where we have no labels $y \in \mathcal{Y}$ but try to find patterns in the input data [9, Sec. 14]. The most prominent example of unsupervised learning is Cluster Analysis, where we try to find clusters of "similar" points in a data set [9, Sec. 14.3]. The notion of similarity may be defined as *(i)* a distance to a common center point, *(ii)* a high distance between points of different clusters, *(iii)* a low variance within the same cluster, and many more. Another (although related) task is *outlier detection*, where we want to detect "unusual" points in a data set, which may be of special interest.

2.3.1. Feature Selection

The complexity of many ML model families scale with the number of input data dimensions, i.e., the number of features. That is, a model with more features requires more memory and more computational effort for its training. Small and fast models are important in applications areas with tight resource constraints, such as embedded systems. Therefore, it is a common goal in ML pipelines to reduce the number of features with minimum information loss by means of a suitable transformation from the raw input data to the training data. This strategy is also known as dimensionality reduction [54].

An important example for such a strategy is FS, where the input dimension is reduced by selecting only a subset of all available features without performing additional preprocessing transformations [55]. This approach is especially effective when dealing with data from sources that produce many redundant or irrelevant features, which can be eliminated without significantly impacting the output quality. For instance, if we try to diagnose a specific disease from a vast array of medical measurements (body temperature, concentration of various substances in a patient's blood, heart rate, etc.), reducing the number of features that are necessary for this diagnosis not only allows for smaller models but might even help experts pin down the underlying cause of the disease. Consequently, FS can be seen as a tool to reduce model complexity and improve ML interpretability in a single step.

Definition 2.10 (Feature Selection). *Presume a classification task on a given data set* $\mathcal{D} = \{(\boldsymbol{x}^i, y_i)\}_{i \in \{1,\dots,N\}}$ *with* n*-dimensional features* $\boldsymbol{x}^i \in \mathcal{X} \subseteq \mathbb{R}^n$ *and class labels* $y_i \in \mathcal{Y} \subseteq \mathbb{N}$ *for all* $i \in \{1,\dots,N\}$. *The problem of FS corresponds to finding a subset* $S \subset \{1,\dots,n\}$ *of these* n *features, such that the reduced data set*

$$\mathcal{D}_S = \{(\boldsymbol{x}^i_S, y_i)\}_{i \in \{1,\dots,n\}}, \text{ where } \boldsymbol{x}_S = (x_j)^\intercal_{j \in S}$$

leads to comparable performance as the original data for some data-driven task, such as classification.

Typically, this subset is found by solving a suitably posed optimization problem, which can also explicitly depend on the classification model.

There are numerous approaches of defining optimal feature subsets and finding or approximating them [55]. Wrapper methods directly use the performance of classification or regression models as a criterion for selecting features [56]. As the model must be retrained on every candidate subset, this method is very resource-intensive. Finding the optimal subset requires a brute-force search of all 2^n possible subsets, which is generally intractable for large feature sets and non-trivial models. Instead, heuristic optimization schemes are often used, such as greedy search or evolutionary algorithms [57, 58].

Filter methods use a measure of relevance, e.g., correlation or Mutual Information (MI) with the label or target variable, for ranking features and discarding those below a certain threshold. While those methods are very easy to compute and work well in practice, they often do not consider redundancy between selected features, leading to subsets that could

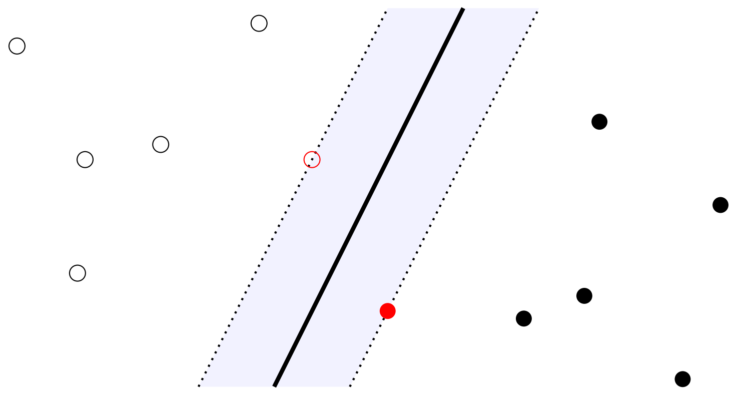


Figure 2.2.: Schematic example of an SVM in 2D space: Circles are data points of two classes (hollow and filled) separated by a hyperplane (here, a line). The margin (light blue area) is defined by two points (red) that lie on its boundary, the *support vectors*.

potentially be much smaller. When considering redundancy, such as pairwise MI between selected features, as an additional criterion to be minimized, the optimization problem's difficulty increases, as it becomes non-linear and NP-hard.

2.3.2. Support Vector Machines

SVMs are among the most extensively studied ML models, developed mainly by Vladimir Vapnik in the 1990s [59, 60]. It is, in its original form, a binary classification model, separating points belonging to one of two classes by means of a separating hyperplane. As there may be infinitely many such hyperplanes, an additional objective is to maximize the *margin*, which is the area around the hyperplane containing no data points, in the hope to achieve the best possible generalization. See Fig. 2.2 for a schematic view of an SVM model. Extensions of this original SVM model allow for multi-class classification and even regression, which we will explore in Chapter 9.

Assume we have a data set $\mathcal{D} = (\boldsymbol{x}^i, y_i) \subset \mathbb{R}^d \times \mathbb{S}$, where y_i denotes the class label, $+1$ or -1. Further assume we have some feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^f$ which transforms the data into some potentially high-dimensional space – for a linear SVM, this feature map is simply the identity function. The separating hyperplane can be represented by its normal vector $\boldsymbol{w} \in \mathbb{R}^f$ and an offset $b \in \mathbb{R}$. To ensure correctness, the optimization requiers that each data point is classified correctly, i.e., each data point should be located on the correct side of the hyperplane:

$$(\boldsymbol{w}^\intercal \varphi(\boldsymbol{x}^i) - b)y_i \geq 1.$$

As the assumption of perfect linear separability is unrealistic for real-world data, each point is assigned a *slack variable* ξ_i, such that a value of $\xi_i > 0$ indicates that $\boldsymbol{x}^i$ violates the correctness condition, i.e.,

$$\xi_i = \max(0, 1 - y_i(\boldsymbol{w}^\intercal \varphi(\boldsymbol{x}^i) - b)). \tag{2.9}$$

Table 2.1.: Examples of kernel functions $K(\boldsymbol{x}, \boldsymbol{x}')$ and their definitions (taken from [61]).

name	*definition*	*parameters*
Linear	$\boldsymbol{x}^\intercal \boldsymbol{x}'$	
Polynomial	$(\boldsymbol{x}^\intercal \boldsymbol{x}' + c)^p$	$p \in \mathbb{N}, c \in \mathbb{R}$
RBF/Gaussian	$\exp\left[-\gamma \lVert \boldsymbol{x} - \boldsymbol{x}' \rVert^2\right]$	$\gamma \in \mathbb{R}_+$

This yields the second objective, which is to minimize the total slack $\sum_i \xi_i$. Combined, we arrive at the following optimization problem.

Definition 2.11 (Primal SVM [9]). *Given a labeled data set $\mathcal{D} = \{(\boldsymbol{x}^i, y_i)\}_{i \in \{1,\dots,N\}} \subset \mathbb{R}^d \times \mathbb{S}$ and a feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^f$, the* primal SVM *is the optimization problem*

$$\begin{aligned} \min_{\boldsymbol{w},b}\ & \frac{1}{2}\lVert \boldsymbol{w} \rVert_2^2 + C\boldsymbol{1}^\intercal \boldsymbol{\xi} \\ \text{s.t. }& (\boldsymbol{w}^\intercal \varphi(\boldsymbol{x}^i) - b) y_i \geq 1 - \xi_i \ \forall\, i \in \{1,\dots,N\}, \end{aligned} \tag{2.10}$$

where $\boldsymbol{\xi} = (\xi_1, \dots, \xi_N)^\intercal$, $\xi_i = \max(0, 1 - y_i(\boldsymbol{w}^\intercal \varphi(\boldsymbol{x}^i) - b))\ \forall\, i \in \{1,\dots,N\}$, and $C > 0$ is a hyperparameter controlling the impact of misclassification.

Dividing Eq. (2.10) by C provides an alternative perspective on SVM training as a hinge-loss minimization with L2-regularization on $\boldsymbol{w}$, where C is the inverse regularization factor. Typically this problem is solved in its Lagrangian dual form.

Definition 2.12 (Dual SVM [9]). *Let $\mathcal{D}$ as in Def. 2.11, with $\boldsymbol{y} \in \mathbb{R}^N$ the vector containing all y_i in $\mathcal{D}$, $\boldsymbol{X}$ the $N \times d$ data matrix as defined in Def. 2.9, and $\boldsymbol{K} = \varphi(\boldsymbol{X})^\intercal \varphi(\boldsymbol{X})$ the* kernel matrix *w.r.t. some feature map φ. The dual form of the SVM training problem given in Def. 2.11 is*

$$\begin{aligned} \max_{\boldsymbol{\alpha}}\ & \boldsymbol{1}_N^\intercal \boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^\intercal (\boldsymbol{y}\boldsymbol{y}^\intercal \odot \boldsymbol{K})\boldsymbol{\alpha} \\ \text{s.t. }& 0 \leq \alpha_i \leq C\ \forall\, i \in \{1,\dots,N\}, \\ & \boldsymbol{\alpha}^\intercal \boldsymbol{y} = 0. \end{aligned}$$

If φ is the identity function, the matrix $\boldsymbol{K} = \boldsymbol{X}^\intercal \boldsymbol{X}$ is called *Gram matrix* and has as entry K_{ij} the inner product of data points $\boldsymbol{x}^i$ and $\boldsymbol{x}^j$ for all $i, j \in \{1,\dots,N\}$. However, many data distributions are not separable by a simple hyperplane in the original feature space. In this case, the feature map may be chosen to project the data into a higher-dimensional space where it is possible to separate them linearly. If $f \gg d$, computing the feature map explicitly may be very expensive. The dual SVM has the great advantage of exploiting the *kernel trick*, as it does not operate on the feature space directly, but instead only uses inner products of feature vectors [9, Sec. 12.3.7]. Therefore, we can define a function $K : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ called *kernel* that implicitly computes $K(\boldsymbol{x}, \boldsymbol{x}') = \varphi(\boldsymbol{x})^\intercal \varphi(\boldsymbol{x}')$ without actually realizing the feature map, allowing for efficient computation of inner products in feature space. The implicit feature spaces of certain kernels, like the RBF kernel, are even infinite-dimensional and cannot be explicitly computed. Common kernels are listed in Table 2.1.

The optimized vector $\boldsymbol{w}$ is the *parameter* or *weight vector* of the SVM model, and b its *bias*. Model predictions are inferred via

$$\hat{y}_{\boldsymbol{w},b}(\boldsymbol{x}) = \operatorname{sign}(\boldsymbol{w}^\intercal \varphi(\boldsymbol{x}) - b), \tag{2.11}$$

where $\operatorname{sign} : \mathbb{R} \to \mathbb{S}$ is the signum function that returns 1 if its argument is greater than 0, and -1 otherwise [9, Eq. 12.2].

2.4. Quantum Computing

Quantum Computing (QC) is a computing paradigm whose origins lie in the latter half of the 20th century. Over the past years, it has gained renewed attention, as physical quantum computers are continually improved, and QC holds the potential to solve certain problems requiring considerable computational efford on *classical* (i.e., non-quantum) computers faster and more efficiently [15, 16]. More recently, QML emerged as a new research field aiming to apply QC techniques to typical ML tasks, such as classification and regression [21].

At its core, QC replaces classical bits with *quantum bits* (or *qubits*, for short) and develops strategies to perform computations with them. In contrast to classical bits, which can take only the values in $\mathbb{B} = \{0, 1\}$, the states of qubits are continuous, assuming discrete values only when measured. The following paragraphs give an overview of the basic concepts of QC. For a more extensive introduction, see [62], from which the information presented in the following paragraphs was taken.

Single-Qubit States The state of a single qubit is denoted by $|\psi\rangle$ (say "ket psi") using the *Bra-ket notation*, which simply stands for a complex-valued column vector $[\alpha_0, \alpha_1]^\intercal$ called *amplitude vector*, with its entries called *amplitudes* accordingly. Qubit state vectors are elements of Hilbert spaces $\mathcal{H}$. For all intents and purposes, we may simply assume that $\mathcal{H} = \mathbb{C}^2$ in the scope of this thesis[2]. The conjugate transpose of $|\psi\rangle$ is notated as $\langle\psi| = (|\psi\rangle)^\dagger = [\alpha_0^*, \alpha_1^*]$ (say "bra psi"). Every amplitude vector $|\psi\rangle$ is normalized to obey

$$\langle\psi|\psi\rangle = |\alpha_0|^2 + |\alpha_1|^2 = 1,$$

where $\langle\phi|\psi\rangle$ is the notation for the inner product between two states. Like classical bits, qubits have two *basis states*, $|0\rangle$ and $|1\rangle$, which are simply the canonical basis vectors of the Hilbert space, $|0\rangle = [1, 0]^\intercal$ and $|1\rangle = [0, 1]^\intercal$, which lets us write $|\psi\rangle = \alpha_0 |0\rangle + \alpha_1 |1\rangle$; this highlights that single-qubit states are a mixture of basis states. If both $\alpha_0 \neq 0$ and $\alpha_1 \neq 0$, the qubit is in *superposition*, which means that it takes both basis states 0 and 1 with a certain probability after measuring [62, Sec. 1.2].

[2]However, the notion of Hilbert spaces is more general in that it allows for infinite-dimensional states.

Measurement It is impossible to directly observe a quantum state, i.e., we cannot measure the amplitudes. Instead, measuring a qubit causes its state to collapse to one of its basis states. The absolute square of each amplitude yields the probability of this qubit to be measured in the respective state. The normalization property of qubit states ensures that the probabilities for each basis state add to 1. The basis states $|0\rangle$ and $|1\rangle$ act like classical bits, always assuming 0 and 1 with probability 1. However, the qubits $|\psi\rangle = [1,1]^\intercal/\sqrt{2}$ and $|\phi\rangle = [1,-\mathrm{i}]^\intercal/\sqrt{2}$ both have an equal probability to be measured in either basis state, as we find

$$\left|\frac{1}{\sqrt{2}}\right|^2 = \left|\frac{-i}{\sqrt{2}}\right|^2 = \frac{1}{2}.$$

Notably, even though the amplitude for basis state $|1\rangle$ is negative (and even complex) in $|\phi\rangle$, the measurement probabilities are identical. This shows that quantum states hold more information than cannot be observed or determined exactly through measurement [62, Sec. 1.2]. Other special 1-qubit states are:

$$|+\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \qquad |-\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$
$$|\rightarrow\rangle = \frac{|0\rangle + \mathrm{i}\,|1\rangle}{\sqrt{2}} \qquad |\leftarrow\rangle = \frac{|0\rangle - \mathrm{i}\,|1\rangle}{\sqrt{2}}.$$

Multi-Qubit States If n qubits $|\phi_1\rangle, |\phi_2\rangle, \dots, |\phi_n\rangle$ form a system, their state can be described by a joint amplitude vector $|\phi\rangle = |\phi_1\phi_2\dots\phi_n\rangle = |\phi_1\rangle \otimes |\phi_2\rangle \otimes \dots \otimes |\phi_n\rangle$, where $\otimes$ denotes the *Kronecker product* described in Section 2.1 [62, Sec. 2.1.7]. This vector has size $N = 2^n$. Just as before, each entry is the amplitude of a basis state $|i\rangle = \boldsymbol{e}_i^n\ \forall i \in \{0,\dots,N-1\}$ of the joint system, which can be interpreted as a binary string of length n. Again, the normalization condition $\langle\phi|\phi\rangle = 1$ holds. Conveniently, the binary vector corresponding to the i-th amplitude and basis state $|i\rangle$ (counting from 0) is the binary expension of i.

Example 2.1. Let $|\psi_1\rangle = |0\rangle = [1,0]^\intercal$, $|\psi_2\rangle = |1\rangle = [0,1]^\intercal$ and $|\psi_3\rangle = |-\rangle = [1,-1]^\intercal/\sqrt{2}$. Their joint state is $|\psi_1\psi_2\psi_3\rangle = |\psi_1\rangle \otimes |\psi_2\rangle \otimes |\psi_3\rangle$, which expands to

$$\frac{1}{\sqrt{2}}\begin{bmatrix} 1\begin{bmatrix} 0\begin{bmatrix}1\\-1\end{bmatrix}\\ 1\begin{bmatrix}1\\-1\end{bmatrix}\end{bmatrix}\\ 0\begin{bmatrix} 0\begin{bmatrix}1\\-1\end{bmatrix}\\ 1\begin{bmatrix}1\\-1\end{bmatrix}\end{bmatrix}\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}0\\0\\1\\-1\\0\\0\\0\\0\end{bmatrix}\begin{array}{l}\texttt{000}\\\texttt{001}\\\texttt{010}\\\texttt{011}\\\texttt{100}\\\texttt{101}\\\texttt{110}\\\texttt{111}\end{array}$$

as a single length 2^3 joint state vector. The gray bit strings on the right indicate the corresponding basis state for each amplitude. As we can see, the probability mass is equally divided between the states 010 and 011, because $|\psi_1\rangle$ and $|\psi_2\rangle$ are in basis states, and only $|\psi_3\rangle$ is in an equal superposition. Therefore every state where $|\psi_1\psi_2\rangle \neq |01\rangle$ has an amplitude of 0.

A multi-qubit state that can be expressed as a Kronecker product of single-qubit states is called *separable* [62, Sec. 1.2.1]. However, through sequences of state manipulations, we can construct quantum states that cannot be decomposed, such as the 2-qubit state $|\psi\rangle = [1, 0, 0, 1]^\intercal$ called *Bell state*: It has an equal probability to be measured as 00 or 11 [62, Eq. 1.7]. We say that the qubits are *entangled*, implying that their individual measurement probabilities are not statistically independent. In the case of the Bell state, the statistical dependence is maximal, as the measurement outcome of the first qubit fully determines the state of the second one. There is no more efficient way to store the amplitudes of arbitrary entangled states than one by one. This is one of the reasons why simulating quantum computing quickly becomes computationally intractable when the number of qubits increases.

Observables If we interpret a quantum state as a probability distribution over its basis states, we may associate some value $g(|x\rangle) \in \mathbb{R}$ with each outcome $|x\rangle$ of a quantum measurement. Now, given a state $|\psi\rangle$ that is in superposition and entangled, we may be interested in the expectation value $\mathbb{E}[g(|x\rangle)]$. To this end, the concept of *observables* provides the formal framework for evaluating expectations over quantum states. An observable $\boldsymbol{O} \in \mathbb{C}^{N\times N}$ (with $N = 2^n$) is a Hermitian matrix, which means that it is equal to its own conjugate transpose, $\boldsymbol{O}^\dagger = \boldsymbol{O}$, and it encodes our evaluation of the quantum state: The expectation of $\boldsymbol{O}$ w.r.t. a quantum state $|\psi\rangle = [\alpha_0, \dots, \alpha_{N-1}]^\intercal$ is given by

$$\langle\psi|\, \boldsymbol{O}\, |\psi\rangle = \sum_{i,j=1}^{N} O_{ij}\alpha_{i-1}\alpha_{j-1}^*. \tag{2.12}$$

If $\boldsymbol{O}$ is diagonal, we obtain the expectation value over all basis vectors $|0\rangle, \dots, |N-1\rangle$, as $\langle\psi|\, \boldsymbol{O}\, |\psi\rangle = \sum_{i=1}^{N} = O_{ii}|\alpha_{i-1}|^2$, and $|\alpha_{i-1}|^2$ is the measurement probability of state $|i-1\rangle$. However, Eq. (2.12) is more general in that it allows for evaluation in arbitrary orthonormal bases of the Hilbert space, i.e., the observable has the form $\boldsymbol{\Lambda}^{-1}\boldsymbol{O}\boldsymbol{\Lambda}$, where $\boldsymbol{\Lambda}$ is any base projection matrix. Intuitively, non-diagonal observables encapsule the action of projecting into a different basis, measuring, and projecting back into the original basis [62, Sec. 2.2.5].

Example 2.2. Assume we have a 3-qubit state $|\psi\rangle = [2, 0, 0, 1, 0, -1, 0, 4]/\sqrt{22}$, and we want to assign to each basis state its number of ones, e.g., $|001\rangle \mapsto 1$, $|111\rangle \mapsto 3$, and so on. The expected number of ones that we observe is given by $\langle\psi|\, \boldsymbol{O}\, |\psi\rangle$ with the observable $\boldsymbol{O} = \text{diag}[0, 1, 1, 2, 1, 2, 2, 3]$, which has the number of 1-bits of all 3-bit strings along its diagonal:

$$\langle\psi|\, \boldsymbol{O}\, |\psi\rangle = \frac{1}{22}(0 + 0 + 0 + 2 + 0 + 2 + 0 + 48) = \frac{52}{22} \approx 2.36.$$

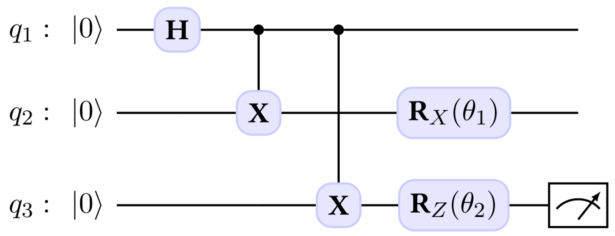


Figure 2.3.: Exemplary quantum circuit with 3 qubits named q_1 to q_3: All qubits are initialized in state $|0\rangle$. First, a Hadamard gate is applied to q_1, and a **CX** gate is applied to both q_2 and q_3, with qubit 1 serving as control. Then, an $\mathbf{R}_X$ gate is applied to q_2, and an $\mathbf{R}_Z$ gate to q_3 with parameters θ_1 and θ_2 respectively. Finally, a measurement of q_3 is taken.

A special set of observables are the *Pauli matrices* denoted by $\boldsymbol{\sigma}^{(x)}$, $\boldsymbol{\sigma}^{(y)}$ and $\boldsymbol{\sigma}^{(z)}$, each with eigenvalues ± 1 [62, Sec. 2.1.3]. Their expectation values w.r.t. a quantum state $|\psi\rangle$ gives the dot product of the quantum state with the corresponding axis of the Bloch sphere. For instance, taking the expectation of $|0\rangle$ w.r.t. $\boldsymbol{\sigma}^{(z)}$ yields $\langle 0| \boldsymbol{\sigma}^{(z)} |0\rangle = 1$, as $|0\rangle$ points along the x-axis. However, $\langle 0| \boldsymbol{\sigma}^{(y)} |0\rangle = 0$, as $|0\rangle$ is perpendicular to the y-axis.

2.4.1. The Circuit Model

So far we have seen what quantum states are and how we can “evaluate” them through Hamiltonians. The question remains how to construct and manipulate quantum states to do something useful with them, i.e., perform quantum computing. A popular approach is through *quantum circuits*, a graphical representation of sequential manipulations of a joint quantum state inspired by logical circuits [62, Sec. 1.3.4].

In Fig. 2.3, an exemplary 3-qubit quantum circuit is shown, which is read from left to right: Initially, all qubits are in state $|0\rangle$, which results in a joint state $|000\rangle = [1, 0, 0, \ldots, 0]^\intercal$. First, an **H**-gate (*Hadamard* gate) is aplied to qubit q_1. Quantum gates, symbolized by rectangular boxes in the circuit diagram, represent linear transformations of the amplitude vector, i.e., complex-valued matrices of size $2^n \times 2^n$ that preserve the property $\langle\phi|\phi\rangle = 1$. Matrices $\boldsymbol{U}$ of this type are called *unitary*, their defining property being $\boldsymbol{U}^\dagger = \boldsymbol{U}^{-1}$ [62, Sec. 1.3.1]. Applying a gate to a single qubit is simply a matrix-vector multiplication, e.g., $\boldsymbol{U}|0\rangle$. The operation performed on the entire joint state is $\mathbf{H} \otimes \boldsymbol{I}_2 \otimes \boldsymbol{I}_2$, as the absence of a gate implies the identity function. The matrix $\mathbf{H}$ representing the Hadamard gate is defined as

$$\mathbf{H} = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}. \tag{2.13}$$

It maps $|0\rangle \mapsto |+\rangle$ and $|1\rangle \mapsto |-\rangle$, creating an equal superposition from basis states. The next operation in Fig. 2.3 is a *controlled NOT* gate (also called **CX** or **CNOT**), which consists of a *control* (here: q_1) and a *target* qubit (here: q_2). It has the effect of invert-

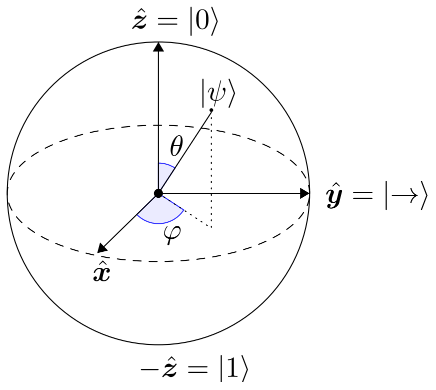

$\hat{\boldsymbol{z}} = |0\rangle$
$|\psi\rangle$
θ
$\hat{\boldsymbol{y}} = |{\rightarrow}\rangle$
φ
$\hat{\boldsymbol{x}}$
$-\hat{\boldsymbol{z}} = |1\rangle$

Figure 2.4.: The Bloch sphere, a 3D representation of a single-qubit quantum state; the basis states w.r.t. the Z-basis are at the north and south pole. Adapted from [62].

ing the target qubit's state if the control qubit is 1, otherwise leaving the target qubit unchanged:

$$\mathbf{CX} = |0\rangle\langle 0| \otimes \boldsymbol{I}_2 + |1\rangle\langle 1| \otimes (1 - \boldsymbol{I}_2) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}. \tag{2.14}$$

Another **CX** gate is applied to q_3 with q_1 as the control. Finally, two *rotation* gates $\mathbf{R}_X$ and $\mathbf{R}_Z$ are applied to q_2 and q_3 with parameters θ_1 and θ_2, respectively. As amplitude vectors have unit length, we can view the set off all single-qubit states as the surface of a unit sphere, which is called the *Bloch sphere*, shown in Fig. 2.4 [62, Sec. 1.2]. Applying $\mathbf{R}_X$ and $\mathbf{R}_Z$ has the effect of rotating a point on this sphere around the X or Z axis by a certain angle, hence $\theta_1, \theta_2 \in [0, 2\pi)$ [62, Sec. 4.2]. An overview of commonly used quantum gates and their corresponding unitary matrices is given in Table 2.2.

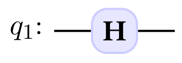
Gates such as $\mathbf{R}_X$, $\mathbf{R}_Y$ and $\mathbf{R}_Z$ provide a way to input classical data into a quantum circuit, forming the basis of Variational Quantum Circuits (VQCs) (see Section 2.4.2). By repeatedly measuring the circuit's final state, i.e., sampling from the probability distribution resulting from the squared amplitudes, and counting the observed basis states, the overall joint distribution can be approximated, which can serve as classical data output. How exactly this information is interpreted depends on the application at hand. Naturally, the more measurements are taken, the closer the empirical distribution approaches the true distribution defined by the qubit state, as less sampling noise is present in the output. However, for each measurement, the quantum state is destroyed, and the entire circuit has to be executed again to take another measurement.

2.4.2. Variational Quantum Circuits

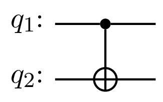
Given a pre-defined quantum circuit containing gates with real-valued parameters (such as $\mathbf{R}_X$, $\mathbf{R}_Y$ and $\mathbf{R}_Z$), we can try to optimize these parameters w.r.t. some objective. This

Table 2.2.: Overview of commonly used quantum gates and their corresponding unitary matrices [62].

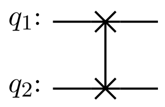
Name	Symbol(s)	#Qubits	Parameters	Unitary
Identity	$\mathbf{I}$	1		$\begin{bmatrix}1 & 0\\0 & 1\end{bmatrix}$
Not/Pauli X	$\mathbf{X}, \boldsymbol{\sigma}^{(x)}$	1		$\begin{bmatrix}0 & 1\\1 & 0\end{bmatrix}$
Pauli Y	$\mathbf{Y}, \boldsymbol{\sigma}^{(y)}$	1		$\begin{bmatrix}0 & -\mathrm{i}\\\mathrm{i} & 0\end{bmatrix}$
Pauli Z	$\mathbf{Z}, \boldsymbol{\sigma}^{(z)}$	1		$\begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}$
Hadamard	$\mathbf{H}$	1		$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$
Phase	$\mathbf{S}$	1		$\begin{bmatrix}1 & 0\\0 & \mathrm{i}\end{bmatrix}$
$\pi/8$	$\mathbf{T}$	1		$\begin{bmatrix}1 & 0\\0 & e^{\mathrm{i}\pi/4}\end{bmatrix}$
Square root of X	$\sqrt{\mathbf{X}}$	1		$\frac{1}{2}\begin{bmatrix}1+\mathrm{i} & 1-\mathrm{i}\\1-\mathrm{i} & 1+\mathrm{i}\end{bmatrix}$
Controlled Not	$\mathbf{CX}, \mathbf{CNOT}$	2		$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\\0 & 0 & 1 & 0\end{bmatrix}$
Swap	$\mathbf{SWAP}$	2		$\begin{bmatrix}1 & 0 & 0 & 0\\0 & 0 & 1 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$
X-Rotation	$\mathbf{R}_X$	1	$\theta \in [0, 2\pi)$	$\begin{bmatrix}\cos(\theta/2) & -\mathrm{i}\sin(\theta/2)\\-\mathrm{i}\sin(\theta/2) & \cos(\theta/2)\end{bmatrix}$
Y-Rotation	$\mathbf{R}_Y$	1	$\theta \in [0, 2\pi)$	$\begin{bmatrix}\cos(\theta/2) & -\sin(\theta/2)\\\sin(\theta/2) & \cos(\theta/2)\end{bmatrix}$
Z-Rotation	$\mathbf{R}_Z$	1	$\theta \in [0, 2\pi)$	$\begin{bmatrix}\exp(-\mathrm{i}\theta/2) & 0\\0 & \exp(\mathrm{i}\theta/2)\end{bmatrix}$



(a) Hadamard gate



(b) CX gate



(c) Swap gate

Figure 2.5.: Graphical representation of quantum gates. **(a)** Single-qubit gates consist of a box with the corresponding symbol inside. **(b)** The Controlled Not gate (**CX**) involves a *control* (q_1) and a *target* qubit (q_2). **(c)** The Swap gate has the effect of swapping the states of two qubits.

framework is named VQCs, and a pre-defined circuit layout is called *ansatz* (plural *ansätze*) [63].

For example, in a quantum chemistry problem, the ansatz might represent the ground-state wavefunction of a molecule, with the parameters corresponding to the amplitudes of different quantum states [64]. In a ML context, the ansatz might represent a classification model, where the parameters are optimized to minimize a loss function over a data set.

The general process of working with VQCs involves *(i)* defining an ansatz based on the structure of the problem, *(ii)* evaluating an objective function (often an expectation value, such as energy or cost) using quantum measurements, and *(iii)* optimizing the parameters classically by minimizing (or maximizing) the objective function. A concrete algorithm is the Variational Quantum Eigensolver (VQE) [65, 66], which uses a hybrid quantum-classical computational approach [67] to minimize the expected energy $\langle \Psi(\boldsymbol{\theta}) | \boldsymbol{H} | \Psi(\boldsymbol{\theta}) \rangle$ of a Hamiltonian $\boldsymbol{H}$. For this purpose, an ansatz $\boldsymbol{U}(\boldsymbol{\theta})$ is prepared on a quantum gate computer such that $|\Psi(\boldsymbol{\theta})\rangle = \boldsymbol{U}(\boldsymbol{\theta}) |0\rangle$. The circuit parameters $\boldsymbol{\theta}$ are learned with a classical optimization routine in order to find an estimate for the ground state $|\Psi\rangle \approx |\Psi(\boldsymbol{\theta})\rangle$. Using an Ising-Hamiltonian, this approach can also be used to solve QUBO problems, as we do in Section 3.4.4.

Choosing the right ansatz is problem-specific, and a poorly chosen ansatz can limit the algorithm's ability to represent the solution space, leading to suboptimal results. Techniques such as hardware-efficient ansätze (which use the native gate set of a quantum processor) or problem-inspired ansätze (which leverage knowledge of the problem structure) are often employed.

2.4.3. Adiabatic Quantum Computing

In recent years, quantum computing has opened up a promising approach to solving QUBO instances, which builds largely on physical properties of the Ising model. As we have seen in Proposition 2.3, we can convert between QUBO instances and Ising models by a simple variable substitution to obtain a Hamiltonian function. By replacing the abstract binary variables σ_i with Pauli operators, we obtain a Hamiltonian operator $\boldsymbol{H}_{\text{Ising}}$, which is an observable of the form

$$\boldsymbol{H}_{\text{Ising}} = \sum_{\substack{i,j=0 \\ i \neq j}}^{n} J_{ij} \boldsymbol{\sigma}_i^{(z)} \boldsymbol{\sigma}_j^{(z)} + \sum_{i=0}^{n} h_i \boldsymbol{\sigma}_i^{(z)} + c, \tag{2.15}$$

where $\boldsymbol{\sigma}_i^{(z)} = \boldsymbol{I}_2^{\otimes i-1} \otimes \boldsymbol{\sigma}^{(z)} \otimes \boldsymbol{I}_2^{\otimes n-i}$ denotes the Pauli-Z operator acting on qubit i. For brevity, we simply write $\boldsymbol{H}$ in the following. Notice that Eq. (2.15) in contrast to Def. 2.8 excludes the terms $J_{ij} \boldsymbol{\sigma}_i^{(z)} \boldsymbol{\sigma}_j^{(z)}$ for the case $i = j$, which is due to the fact that a 2-qubit operator cannot be applied to the same qubit. Assuming w.l.o.g. that $\boldsymbol{J}$ is a hollow matrix resolves this disparity.

The Pauli-Z operator has eigenvalues ± 1 with corresponding eigenvectors $|0\rangle$ and $|1\rangle$. Since the Pauli spin matrices of different qubits commute, the minimum eigenstate $|\Psi\rangle$ of the Hamiltonian with $\boldsymbol{H}|\Psi\rangle = E|\Psi\rangle$ can be written in terms of $|\Psi\rangle = |\psi_1\rangle \otimes \cdots \otimes |\psi_n\rangle$, where $|\psi_i\rangle \in \{|0\rangle, |1\rangle\} \, \forall \, i \in \{1, \dots, n\}$. Intuitively, if qubit i is in state $|0\rangle$, the eigenvalue of $h_i \boldsymbol{\sigma}_i^{(z)}$ is $-h_i$, and if it is in state $|1\rangle$, the eigenvalue is $+h_i$ – or we can conveniently say that the eigenvalue of $h_i \boldsymbol{\sigma}_i^{(z)}$ is simply $h_i \sigma_i$, where $\sigma_i \in \mathbb{S}$ is the abstract binary variable of the original Ising model, obtained by substituting $|1\rangle \mapsto -1$ and $|0\rangle \mapsto +1$. Similarly, the eigenvalues of $J_{ij} \boldsymbol{\sigma}_i^{(z)} \boldsymbol{\sigma}_j^{(z)}$ come out to be exactly $J_{ij}\sigma_i\sigma_j$ with $\sigma_i, \sigma_j \in \mathbb{S}$ defined as above.

As a consequence of $\boldsymbol{\sigma}^{(z)}$ being diagonal, $\boldsymbol{H} \in \mathbb{C}^{2^n \times 2^n}$ is also a diagonal observable, containing the energy values of all possible states $\boldsymbol{\sigma} \in \mathbb{S}^n$ of the Ising model in lexicographical order. Therefore, the minimal eigenvalue E represents the minimal energy value of the corresponding QUBO instance, $f^*(\boldsymbol{Q})$. The respective solution vector $\boldsymbol{z}^* \in S^*(\boldsymbol{Q})$ can be obtained from the eigenstate $|\Psi\rangle$ with the assignment $z_i^* = |\langle \boldsymbol{e}_i^n | \Psi | \boldsymbol{e}_i^n | \Psi \rangle|^2$ based on a projective measurement of each qubit. Summarized, the QUBO problem can be transformed into the problem of finding the minimum eigenstate (or ground state) $|\Psi\rangle$ of $\boldsymbol{H}$.

However, finding the ground state of a Hamiltonian is in general also a non-trivial problem, and possible solution strategies depend on the properties of the quantum hardware and the shape of $\boldsymbol{H}$. The most common approach to this problem is Quantum Annealing (QA) [17, 68], which is suitable for special-purpose quantum computing devices named "quantum annealers". The Canadian company D-Wave has specialized on building quantum annealers, whose flagship device at the time of this writing has around 5,000 physical qubits[3].

QA exploits the adiabatic theorem [69] by preparing the ground state $|\Psi_0\rangle$ of a simple "mixing Hamiltonian" $\boldsymbol{H}_0$ and then slowly transferring the system into the ground state $|\Psi\rangle$ of the target Hamiltonian $\boldsymbol{H}_1$ through adiabatic time evolution [70], which can be modelled as

$$\boldsymbol{H}(s) = (1 - A(s))\boldsymbol{H}_0 + A(s)\boldsymbol{H}_1,$$

where $A : [0, 1] \mapsto [0, 1]$ is some monotonic interpolation schedule with $A(0) = 0$ and $A(1) = 1$, and s slowly transitions from 0 to 1 over a certain time time period T. A typical choice of $\boldsymbol{H}_0$ is $-\sum_{i=1}^{n} \boldsymbol{\sigma}_i^{(x)}$, whose ground state is $|+\rangle^{\otimes n} = \sum_{x \in \mathbb{B}^n} |x\rangle / 2^{N/2}$, an equal superposition of all qubits as can be generated through parallel Hadamard gates $\mathbf{H}^{\otimes n} |0 \dots 0\rangle$. The adiabatic theorem states that, if the transition is performed slowly enough, the system will remain in its ground state, which means that at the end the ground state of the target Hamiltonian $\boldsymbol{H}_1$ can be sampled by measurement.

Limitations The rate at which the Ising Hamiltonian can be changed without leaving the ground state is strongly influenced by the *minimal spectral gap* of $\boldsymbol{H}(s)$, which is the

[3]Descriptions of their hardware systems can be found at `www.dwavesys.com` (last retrieved on June 3, 2025)

distance betwesen its lowest and second-to lowest eigenvalues across $s \in [0, 1]$ denoted by $\gamma_{\boldsymbol{H}}$. More precisely, the required annealing time T to preserve the ground state with high probability obeys

$$T \propto \frac{1}{\gamma_{\boldsymbol{H}}^2},$$

as follows from the adiabatic theorem [69]. Ising models encoding certain NP-hard problems have been shown to produce exponentially small minimal spectral gaps [71]. Generally, the minimal spectral gap is influenced by the structure of the problem that the Ising model encodes [72], which is subject of ongoing research. Making any assumptions about the spectral gap, or even computing it, is notoriously hard [73].

The physical layout of QA devices is another limiting factor: D-Wave annealers use a special connectivity structure for their qubits, which contains only a subset of all possible couplings between qubit pairs. Their most recent topology is called Zephyr, allowing for up to 82 densely connected qubits on their Advantage 2 prototype[4]. A heuristic hybrid approach can be realized by splitting the initial QUBO instance into smaller sub-problems, solving each with QA, and merging the sub-results, which allows for larger problem sizes at the cost of higher running time and weaker optimality guarantees (see, e.g., [74]). Integrated Control Error (ICE) is another phenomenon that distorts the Ising parameters on the QA chip, resulting in additional noise depending on the qubits' physical layout and connectivity [75].

2.4.4. The NISQ Era

At the time of writing, QC devices is still in a phase commonly known as the Noisy Intermediate-Scale Quantum (NISQ) era. This term, coined by Preskill [20], describes the situation where quantum devices with hundreds to thousands of qubits are becoming available, but these qubits are still noisy, meaning they are prone to errors due to imperfections in hardware, control systems, and environmental factors. Moreover, full-scale quantum error correction is not yet feasible due to the resource demands of encoding logical qubits using many physical qubits [76]. As a result, NISQ devices operate in a regime between classical computation and future fault-tolerant quantum computers.

NISQ devices typically have a few dozen to a few hundred qubits, which is far from the millions of qubits needed for robust fault-tolerant quantum computing. These qubits do allow for significant quantum parallelism, enabling certain quantum algorithms to be run at scales beyond classical simulation [77]. However, qubits and gate operations suffer from noise, as qubit coherence times are limited, and gate fidelities are far from perfect, leading to errors in quantum operations. The high level of noise and relatively short coherence times limit the depth and complexity of quantum circuits that can be reliably executed [20]. The depth of a quantum circuit is the number of consecutive quantum gate operations. Error mitigation techniques, such as zero-noise extrapolation [78] and

[4] `https://www.dwavesys.com/media/2uznec4s/14-1056a-a_zephyr_topology_of_d-wave_quantum_processors.pdf` (last accessed June 3, 2025)

error-resilient algorithms, are being developed, but they are not a substitute for full error correction. As a result, circuits need to be carefully designed to minimize depth (the number of sequential gate operations) and avoid excessive error accumulation.

One of the defining features of the NISQ era is the use of hybrid quantum-classical algorithms, where a quantum computer performs parts of a computation while a classical computer assists in areas like parameter optimization or data post-processing. Examples of this are variational algorithms like VQE discussed in Section 2.4.2 [67, 65]. Assessing the true performance of NISQ devices is difficult because their computational advantage is often problem-specific, and their quantum advantage may not translate into practical utility for real-world problems yet.

2.4.5. Working with Quantum Hardware

Despite numerous persisting challenges discussed before, quantum computers have become more and more accessible over the past decade. In 2016, IBM launched a cloud-based service, which allows members of the public to access their quantum devices and execute quantum circuits on them[5]. In 2024, IBM provided access to 12 devices, each with a minimum of 127 physical qubits[6]. In addition, they maintain a Python software package named *Qiskit* [79], which allows users to compile quantum circuits and upload them to a quantum *backend* (i.e., one of the quantum devices), which executes them and sends the measurement results back. To synchronize access across many users, Qiskit bundles a user's circuits into *jobs*. If the target backend is busy, jobs wait in a queue until all preceding jobs have been processed. The number of circuits in a single job, number of measurements, and the circuit depth is limited. If a backend's workload happens to be very high, performing experiments may take a very long time due to queue waiting times. Qiskit also provides simulators, which perform the unitary operations of a given circuit classically. Naturally, this is only feasible for circuits involving a small number of qubits.

While there is a theoretically infinite number of unitary operations, real quantum computers can only realize a very small number of quantum gates. This *native* set of gates differs across QC architectures. Any unitary operation outside of this native gate set has to be *factorized* into a sequence of gates that the quantum device can realize, which typically leads to an increase of the number of gates by a factor of around 4 to 5 [80]. A set of gates that can be provably used to construct all possible unitaries is called *universal*, such as the set $\{\mathbf{H}, \mathbf{S}, \mathbf{CX}, \mathbf{T}\}$ [62, Sec. 4.5]. An important theoretical result in this context is the *Solovay-Kitaev Theorem*, which implies that any single-qubit gate can be approximated to accuracy ϵ using only $\mathcal{O}(\log^c(1/\epsilon))$ gates from this small set, where $c \approx 2$ [62, Sec. 4.5.3]. A universal quantum gate computer needs to have a native gate set that is universal.

[5]`https://www.ibm.com/quantum/blog/quantum-five-years(lastaccessedJune3,2025)`
[6]`https://quantum.ibm.com/services/resources` (last accessed June 3, 2025)

The process of preparing an entire quantum circuit for execution on a particular quantum device is called *transpilation* and involves, among a few other things, *(i)* converting multi-qubit gates into single-qubit and two-qubit gates, *(ii)* assigning logical qubits to physical qubits, *(iii)* converting gates into the native gate set of the target device, and *(iv)* optimizing the circuit (e.g., removing redundant gates)[7].

Similar to the IBM cloud service, the company D-Wave offers remote access to their quantum annealers via their platform D-Wave Leap[8]. Unlike GQC devices, quantum annealers are much less configurable and only solve Ising models, which are fully specified by the parameters $\boldsymbol{J}$ and $\boldsymbol{h}$. QUBO problems are converted to their corresponding Ising models according to Proposition 2.3. Certain parameters, like the annealing time T and the number of final read, can be set manually[9].

2.4.6. State of the Art

While all challenges and limitations persists, QC research has nonetheless produced a range of promising results, which shall be discussed briefly in the following. In December 2024, Google presented a state-of-the-art quantum chip named Willow, which performed a benchmark computation in under 5 minutes. In contrast, today's fastest supercomputers would need around 10^{25} years to do the same[10]. At the same time, Google claimed that Willow achieves the lowest qubit error rate, which constitutes the step in the direction of large-scale error-corrected QC [81].

QC is used in the simulation of many-body systems in physics [82]. Currently available hardware is able to simulate systems approaching 100 qubits. While there has been significant progress towards achieving a quantum advantage, the problems of noise, gate errors, and short decoherence times remain as the biggest limiting factors.

In QML research, there is an ongoing race for developing quantum counterparts of classical ML methods [24]. Examples include, but are by no means limited to, quantum Boltzmann machines [83], hybrid quantum autoencoders [84], quantum neural networks [85], including routines of classical-quantum transfer learning [86], quantum deep convolutional neural networks [87], and long short-term memory quantum neural networks [88]. A popular target domain is image classification using popular benchmark data sets. As a general trend, research seems to focus on emulating classical ML models and applications. The results achieved by these approaches is often only only as good as corresponding classical results, at best [24], and the methods are, again, limited in scale.

Critical voices question the usefulness of classical benchmarks for assessing the performance of quantum methods: For instance, many popular classical benchmarks have too many features and are compressed or reduced in complexity to make them suitable for

[7]`https://docs.quantum.ibm.com/guides/transpile` (last accessed June 3, 2025)
[8]`https://cloud.dwavesys.com/leap` (last accessed June 3, 2025)
[9]`https://docs.dwavesys.com/docs/latest/c_solver_parameters.html` (last accessed June 3, 2025)
[10]`https://blog.google/technology/research/google-willow-quantum-chip/` (last accessed June 3, 2025)

quantum methods, going against the intended purpose of a benchmark [89]. It was shown that, in an unbiased evaluation, classical methods that are applied out-of-the-box consistently outperform corresponding quantum models, and that strongly entangling ansätze do not significantly improve QML model performance [89]. All of this suggests that QC applications have a long way ahead until they are truly competitive, and that copying classical methods in the hopes of immediately achieving some kind of quantum advantage is not a promising path to follow.

Part II.

Quantum Computing for Classical Machine Learning

3. Feature Selection on Quantum Annealers

While from a practical point of view still in its infancy, QC is a promising future technology with some strange and unintuitive properties, holding the potential to efficiently perform computations in exponentially large spaces and speed up algorithms. However, the natural question arising is *how* we can use these quantum properties to do useful things. This part of the thesis attempts to give an (albeit necessarily incomplete) answer to the following question: **How can QC supplement classical methods and algorithms?**

Naturally, there are thousands of scientific papers about applications of QC to various problems across academia. In the context of this thesis, we focus on ML methods to which we apply QC. To this end, we pick two steps from the ML *pipeline* shown in Fig. 2.1 and demonstrate how QC can be included as a powerful building block, achieving some kind of advantage over purely classical methods. More precisely, in this chapter we will look at FS, an important pre-processing step, and how we can use AQC to find the optimal selection within the exponential space of feature subsets.

In Chapter 4, we take a closer look at a particular classification model, the SVM, and show how both QC paradigms discussed in Section 2.4 can be applied, once for training and once for deployment. Taken together, these two different applications will serve to give an impression of how QC can be incorporated into an ML workflow. While the scale at which QC is applicable today is severely limited by the noisy and imperfect devices that are accessible to us, considering the quick advances in QC hardware, we can expect the methods presented here to become more practically relevant in the future.

This chapter is based on the publication [5]. The author of this thesis developed the idea of using mutual information to perform feature selection using Qubo. He devised the Qubo formulation, the method to control the feature subset size and its correctness proof, and the final QFS algorithm. Further, he designed and conducted all experiments and wrote the majority of the paper.

As we have seen in Section 2.3.1, FS is an important pre-processing step in ML pipelines, reducing the number of features and, consequently, the model size and complexity. This, in turn, makes models both easier to train and deploy, and more interpretable for human users. At its core, FS is an optimization problem with an exponentially growing search space: Given n features, the number of possible subsets is 2^n.

As shown in Def. 2.3, subsets of n-element sets are isomorphic to binary vectors, which allows us to view FS as an optimization problem over the set $\mathbb{B}^n$. If we manage to formulate the objective function of the FS optimization problem as a quadratic function, we can develop a QUBO formulation that can be solved using quantum annealing. While model-dependent objective functions that may even involve re-training a model (as is the case for Wrapper methods) cannot be captured faithfully by a quadratic function, filter methods based on statistical properties of the input data are suitable candidates for QUBO.

In the following sections we describe an algorithm for selecting a specific number of features using a QUBO formulation that can be applied to any combination of statistical *redundancy* and *importance* measures. To this end, we describe in Section 3.1 how a weight matrix $\boldsymbol{Q}$ can be computed whose optimal solution encodes a feature subset based on a trade-off between importance and redundancy. In order to control the number of features in our solution, in Section 3.3 we develop an iterative procedure around our QUBO formulation that finds the correct balancing factor between importance and redundancy to produce the desired result. We show by proof that a solution exists for every desired number of features. Finally, we benchmark the algorithm on different data sets using both classical and actual quantum hardware to demonstrate its effectiveness.

3.1. QUBO-Based Feature Selection

Recall our definition of FS given in Def. 2.10, where we have a data set $\mathcal{D} = \{(\boldsymbol{x}^i, y_i)\}$ for $i \in \{1, \dots, N\}$ with n-dimensional features $\boldsymbol{x}^i \subset \mathbb{R}^n$ and class labels $y_i \subset \{1, \dots, C\}$ for all $i \in \{1, \dots, N\}$.

For a given subset of features, we want to *(i)* maximize the importance of each single feature for the prediction, and simultaneously *(ii)* minimize the amount of redundancy between each pair of features. This way, we obtain a subset that contains both important features, but is not unnecessarily large by including features whose informative value is already covered by other features.

Our main information-theoretic tool is Mutual Information (MI), which measures the amount of information shared between two random variables.

Definition 3.1 (Mutual Information [90]). *Let X, Y be two random variables over two spaces $\mathcal{X}$ and $\mathcal{Y}$ respectively. Further, let $p_X : \mathcal{X} \to \mathbb{R}_{0+}$ and $p_Y : \mathcal{Y} \to \mathbb{R}_{0+}$ their probability mass functions, and $p_{X,Y} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_{0+}$ their joint probability mass function. The* mutual information *between X and Y is defined as*

$$\mathrm{I}(X;Y) = D_{KL}(p_{X,Y} \,\|\, p_X \cdot p_Y) = \int_{\mathcal{Y}} \int_{\mathcal{X}} p_{X,Y}(x,y) \log\left(\frac{p_{X,Y}(x,y)}{p_X(x) p_Y(y)}\right) \mathrm{d}x \, \mathrm{d}y,$$

where $D_{KL}(\cdot\|\cdot)$ is the Kullback-Leibler divergence, and $p_X \cdot p_Y$ is the product distribution defined by $(p_X \cdot p_Y)(x,y) = p_X(x) \cdot p_Y(y)$.

Discretizing the Data The calculation of mutual information requires explicit knowledge about the joint probability mass function of features and labels and the corresponding marginals, which are in general difficult to estimate empirically for real-valued data, as we have to impose an underlying distribution family. Therefore, we map all available feature values from the data set $\mathcal{D}$ into B discrete bins. Specifically, for each separate feature dimension i we take all $\ell/(B+1)$-quantiles for $\ell \in \{0,\dots,B\}$, which we denote by q_i^ℓ. With these, we define bins $\mathcal{B}_i^\ell$ as intervals $[q_i^{\ell-1}, q_i^\ell)$ for $\ell \in [B-1]$, and $\mathcal{B}_i^B = [q_i^{B-1}, q_i^B]$. Finally, we set $b_i^j = \ell$ for the single ℓ that fulfills $x_i^j \in \mathcal{B}_i^\ell$. Since the labels are discrete by definition, no binning is necessary in $\mathcal{Y}$. This way, we obtain a discretized data set $\hat{\mathcal{D}} = \{(\boldsymbol{b}^i, y^i)\}_{i\in\{1,\dots,N\}}$ with $b_i^j \in \{1,\dots,B\}$ for all $j \in \{1,\dots,N\}$ and $i \in \{1,\dots,n\}$.

The empirical probability mass function after discretization reads

$$\hat{p}(\boldsymbol{b}, y) = \frac{1}{N} \sum_{(\boldsymbol{b}',y')\in\hat{\mathcal{D}}} \mathbb{1}\{\boldsymbol{b} = \boldsymbol{b}' \wedge y = y'\}. \tag{3.1}$$

Consequently, we can approximate the information entropies as

$$\mathrm{I}(x_i; y) \approx \sum_{b\in\{1,\dots,B\}} \sum_{y\in\mathcal{Y}} \hat{p}_{X_i,Y}(b,y) \log\left(\frac{\hat{p}_{X_i,Y}(b,y)}{\hat{p}_{X_i}(b)\hat{p}_Y(y)}\right) \tag{3.2}$$

$$\mathrm{I}(x_i; x_j) \approx \sum_{b\in\{1,\dots,B\}} \sum_{b'\in\{1,\dots,B\}} \hat{p}_{X_i,X_j}(b,b') \log\left(\frac{\hat{p}_{X_i,X_j}(b,b')}{\hat{p}_{X_i}(b)\hat{p}_{X_j}(b')}\right), \tag{3.3}$$

where we make use of the marginal distributions of the probability mass functions of feature subsets and labels. These can be computed simply by counting, which yields

$$\hat{p}_{X_i,X_j}(b_i,b_j) = \sum_{\substack{y\in\mathcal{Y},\\ b_k\in\{1,\dots,B\}\forall k\neq i,j}} \hat{p}(\boldsymbol{b},y), \tag{3.4}$$

$$\hat{p}_{X_i,Y}(b_i,y) = \sum_{b_k\in\{1,\dots,B\}\forall k\neq i} \hat{p}(\boldsymbol{b},y), \tag{3.5}$$

$$\hat{p}_{X_i}(b_i) = \sum_{\substack{y\in\mathcal{Y},\\ b_k\in\{1,\dots,B\}\forall k\neq i}} \hat{p}(\boldsymbol{b},y), \tag{3.6}$$

$$\hat{p}_Y(y) = \sum_{b_k\in\{1,\dots,B\}\forall k} \hat{p}(\boldsymbol{b},y). \tag{3.7}$$

Discretizing allows us to approximate the MI values while greatly simplifying the estimation procedure, as no assumption on the underlying probability distribution of the data is required. Additionally, the estimation is consistent, which means when the number of bins approaches infinity the true MI between continuous features will be recovered [91, Theorem 3.2]. Using these tools, we compute an importance vector $\boldsymbol{I}$ and a redundancy matrix $\boldsymbol{R}$ from our (discretized) data.

Importance The importance vector $\boldsymbol{I} \in \mathbb{R}^n_{0+}$ contains the elements

$$I_i = \mathrm{I}(x_i; y) \geq 0 \;\forall i \in \{1, \dots, n\}, \tag{3.8}$$

and represents the MI between each separate feature and the class label y. As we wish to retain as much information about the label as possible in order to perform classification, it is a useful measure of importance for the purpose of FS. As the QUBO energy is minimized, importance appears with a negative sign in Eq. (3.10).

Redundancy The pairwise redundancy matrix $\boldsymbol{R} \in \mathbb{R}^{n \times n}_{0+}$ contains the elements

$$R_{ij} = \mathrm{I}(x_i; x_j) \geq 0 \;\forall i, j \in \{1, \dots, n\}, \tag{3.9}$$

which is symmetric and positive semidefinite by definition. The resulting matrix collects the mutual information between each pair of individual features, and therefore measures their redundancy, as a high value suggests a large overlap in their information content. We set $R_{ii} = 0$ for all $i \in \{1, \dots, n\}$, as we wish to ignore each feature's redundancy with itself. As we want to minimize redundancy, $\boldsymbol{R}$ appears with a positive sign in Eq. (3.10). To simplify our notation, we omit the dependence of R_{ij} and I_i on the input data and discretization procedure, however, it is important to carefully cosider the discretization method as a free parameter.

QUBO Formulation Finally, to arrive at a QUBO formulation, we define

$$\begin{aligned} Q(\boldsymbol{z}; \alpha) &= -\alpha \sum_{i=1}^{n} I_i z_i + (1-\alpha) \sum_{i,j=1}^{n} R_{ij} z_i z_j \\ &= -\alpha \boldsymbol{z}^\intercal \boldsymbol{I} + (1-\alpha) \boldsymbol{z}^\intercal \boldsymbol{R} \boldsymbol{z} \\ &= \boldsymbol{z}^\intercal \boldsymbol{Q}(\alpha) \boldsymbol{z}, \end{aligned} \tag{3.10}$$

where the user-defined parameter $\alpha \in [0, 1]$ balances the influence of importance versus redundancy. The weight matrix $\boldsymbol{Q}(\alpha)$ can be defined as $\boldsymbol{Q}(\alpha) = (1-\alpha)\boldsymbol{R} - \alpha\,\mathrm{diag}[\boldsymbol{I}]$, whose elements read

$$Q_{ij}(\alpha) = (1-\alpha) R_{ij} - \alpha\, \mathbb{1}\{i = j\} I_i. \tag{3.11}$$

The solution of this QUBO instance represents the optimal subset of features: From the solution bit vector $\boldsymbol{z}^*$ we can obtain the set of feature indices through $\iota_n(\boldsymbol{z}^*)$ (see Def. 2.3). See Fig. 3.1 for a schematic view of the entire FS process using this method.

3.2. Feature Subset Size

Typically in FS, we want to control the number of features we want to select. We can enforce this by adding a constraint to Eq. (3.10), penalizing any number of result bits

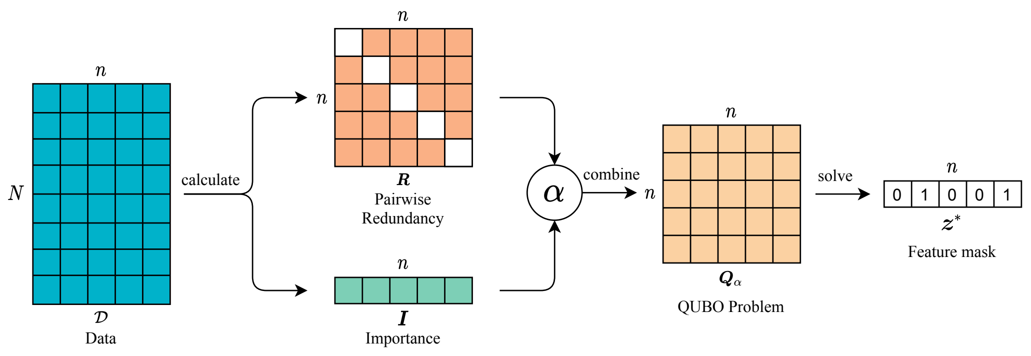


Figure 3.1.: QFS pipeline: Redundancy $\boldsymbol{R}$ and importance $\boldsymbol{I}$ are computed from a discrete data set (see Eqs. (3.8) and (3.9)). Both metrics are balanced with a factor α, yielding a QUBO matrix (Eq. (3.10)) whose minimizing binary vector $\boldsymbol{z}^*$ indicates the feature selection. Source: [5].

different from k. To this end, we could, in theory, add a penalty term $\lambda\left(\left(\sum_i z_i\right) - k\right)^2$ with $\lambda > 0$ to $Q(\boldsymbol{z}, \alpha)$, which is only equal to zero for a selection of exactly k features (compare Section 2.2). The problem can then be solved to obtain a solution with $k = \|\boldsymbol{z}^*\|_1$.

However, two challenges arise when using this approach. The suitable choice of λ is not obvious and depends on the magnitude of $Q(\boldsymbol{z}, \alpha)$: If chosen too small, the imposed constraint might be ignored for certain solutions, leading to constraint violations. In contrast, if chosen too large, it may lead to a very large value range and, consequently, to loss of precision, as having both very small and very large elements in the weight matrix limits the amount of scaling to amplify meaningful differences between loss values — this problem is discussed in much more detail in Chapter 6. Moreover, it is not clear whether a feasible solution to the constrained problem can be found at all in the first place.

We employ an alternative strategy to specify the number of selected features. Instead of using penalty terms, we exploit the parameter α, which by balancing the influence of importance and redundancy influence the number of 1-bits in the solution. This can be seen by considering the smallest and largest possible values of $\alpha \in [0, 1]$: If we set $\alpha = 0$, we put full emphasis on redundancy, and all diagonal entries of $\boldsymbol{Q}(0)$ become 0. Naturally, both the empty feature set ($\boldsymbol{z}^* = \boldsymbol{0}^n$) as well as any single feature ($\boldsymbol{z}^* = \boldsymbol{e}_i^n\, \forall i \in \{1, \dots, n\}$) is least redundant. Conversely, if $\alpha = 1$, the resulting problem is linear, and all coefficients are linear, so that all features ($\boldsymbol{z}^* = \boldsymbol{1}^n$) are selected as the optimal solution (bit vector $\boldsymbol{1}^n$). We observe that by varying α from 0 to 1 the number of bits $\|\boldsymbol{z}^*\|_1$ increases monotonically in steps of one from 0 to n. This observation suggests that there is an α value that results in a subset of size k for any desired $k \in \{0, 1, \dots, n\}$. Proposition 3.1 proves that this is indeed true.

Proposition 3.1 (Existance of α [5]). For all $Q(\cdot, \alpha)$ defined as in Eq. (3.10) and $k \in \{0, 1, \dots, n\}$, there is an $\alpha \in [0, 1]$ such that $\boldsymbol{z}^* \in S^*(\boldsymbol{Q}(\alpha))$ and $\|\boldsymbol{z}^*\|_1 = k$.

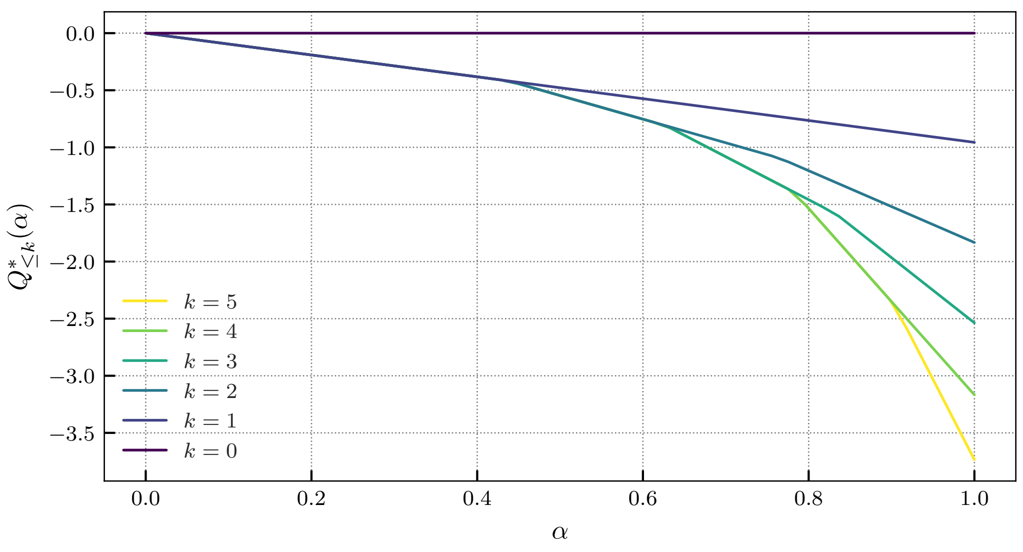


Figure 3.2.: Graphs of $Q^*_{\leq k}$ used in the proof of Proposition 3.1, for $k \in \{0, \dots, n\}$ with $n = 5$. Redundancy and importance values were randomly sampled. Source: [5].

Proof. For ease of notation, we define $I(\boldsymbol{z}) = \boldsymbol{I}^\intercal \boldsymbol{z}$ and $R(\boldsymbol{z}) = \boldsymbol{z}^\intercal \boldsymbol{R} \boldsymbol{z}$ just in the scope of this proof, where $\boldsymbol{I}$ and $\boldsymbol{R}$ are defined as in Eqs. (3.8) and (3.9), computed from an arbitrary but fixed data set with n features. Recall that $R(\boldsymbol{z}) \geq 0$ and $I(\boldsymbol{z}) \geq 0$ for all $\boldsymbol{z} \in \mathbb{B}^n$ due to non-negativity of mutual information. Firstly, note that for $\alpha = 0$ both the zero vector $\mathbf{0}^n$ and all unit vectors $\boldsymbol{e}_i^n$ are optimal w.r.t. $Q(\cdot, 0)$, as $Q(\mathbf{0}, 0) = 0$ and $\forall i \in \{1, \dots, n\} : \ Q(\boldsymbol{e}_i, 0) = R_{ii} = 0$ by definition of $\boldsymbol{R}$. This covers the cases $k = 0$ and $k = 1$. Further, if $\alpha = 1$ we have $Q(\boldsymbol{z}, 1) = -I(\boldsymbol{z}) = -\sum_{i \in \{1,\dots,n\}} I_i z_i$, which is trivially minimized by the one vector $\mathbf{1}^n$, covering the case $k = n$. Now, for all $k \in \{0, \dots, n\}$, consider the functions

$$Q^*_{\leq k}(\alpha) = \min_{\boldsymbol{z} \in \mathbb{B}^n} Q_\alpha(\boldsymbol{z}) \text{ s.t. } \|\boldsymbol{z}\|_1 \leq k. \tag{3.12}$$

These functions are piece-wise linear and strictly decreasing in α (see Figure 3.2), due to $\partial Q_\alpha(\boldsymbol{z}) / \partial \alpha = -(R(\boldsymbol{z}) + I(\boldsymbol{z})) \leq 0$ and non-negativity of $R(\boldsymbol{z})$ and $I(\boldsymbol{z})$. This implies further for all $\alpha \in [0, 1]$ and $k \in \{1, \dots, n\}$ that

$$\min_{\substack{\boldsymbol{z} \in \mathbb{B}^n \\ \|\boldsymbol{z}\|_1 \leq k-1}} R(x) \leq \min_{\substack{\boldsymbol{z} \in \mathbb{B}^n \\ \|\boldsymbol{z}\|_1 \leq k}} R(x) \tag{3.13}$$

$$\max_{\substack{\boldsymbol{z} \in \mathbb{B}^n \\ \|\boldsymbol{z}\|_1 \leq k-1}} I(x) \leq \max_{\substack{\boldsymbol{z} \in \mathbb{B}^n \\ \|\boldsymbol{z}\|_1 \leq k}} I(x). \tag{3.14}$$

This immediately implies that for any $k < k'$

$$Q^*_{\leq k}(0) \leq Q^*_{\leq k'}(0) \tag{3.15}$$

$$Q^*_{\leq k}(1) \geq Q^*_{\leq k'}(1) \ , \tag{3.16}$$

which further implies that, unless $Q^*_{\leq k}$ and $Q^*_{\leq k'}$ are equal, there is at least one point β such that for all $\alpha' > \beta$ we have $Q^*_{\leq k'}(\alpha') \leq Q^*_{\leq k}(\alpha')$ as a consequence of $Q^*_{\leq k}$ and $Q^*_{\leq k'}$ being non-increasing, from which follows the proof. If indeed $Q^*_{\leq k}$ and $Q^*_{\leq k'}$ were equal, both binary vectors $\boldsymbol{z}$ and $\boldsymbol{z}'$ with $\|\boldsymbol{z}\|_1 = k$ and $\|\boldsymbol{z}'\|_1 = k'$ would be optimal, from which the proof still follows. □

This result shows that additional constraints on the QUBO instance are not required to control the number of features, as there all suitable choices of α for every possible k. The problem that remains is finding the specific value of α.

3.3. The QFS Algorithm

Proposition 3.1 lets us devise an algorithm that, given pre-computed $\boldsymbol{R}$, $\boldsymbol{I}$ and k, returns an α^* such that an optimal feature subset vector $\boldsymbol{z} \in Q^*(\alpha^*)$ has exactly k non-zero entries. For this purpose, we introduce

$$Q^*_k(\alpha) = \min_{\substack{\boldsymbol{z}\in\{0,1\}^n \\ \text{s.t. } \|\boldsymbol{z}\|_1 = k}} Q(\boldsymbol{z}, \alpha) \tag{3.17}$$

with $0 \leq k \leq n$, i.e., the minimal function value of $Q(\boldsymbol{z}, \alpha)$ for a given α when the number of ones in the solution is restricted to k. We denote the minimal energy wrt. k (i.e., the globally minimal energy) by

$$Q^*(\alpha) = \min_k Q^*_k(\alpha). \tag{3.18}$$

If we have an oracle for $Q^*(\alpha)$ that returns a global optimum for any $Q(\cdot, \alpha)$, an appropriate value for α^* can be found in $\mathcal{O}(\log n)$ steps through binary search. Note that the value of α^* is not necessarily unique.

As an additional step that proved to be practically relevant, we introduce a threshold $\epsilon \geq 0$, such that $Q_{ii}(\alpha)$ as defined in Eq. (3.11) is set to some small positive value $\mu > 0$ if $\alpha I_i < \epsilon$ for all $i \in \{1, \dots, n\}$. That is, we change $Q_{ij}(\alpha)$ to $Q_{ij}(\alpha, \epsilon, \mu)$ with

$$Q_{ij}(\alpha, \epsilon, \mu) = \begin{cases} \mu & \text{if } i = j \wedge \alpha I_i < \epsilon \\ Q_{ij}(\alpha) & \text{otherwise} \end{cases} \tag{3.19}$$

for arbitrary but constant values of ϵ and μ. We observed that importance values very close to zero have virtually no influence on the function value, and the solvers tended to include or exclude the concerning features randomly. As we want to minimize the number of necessary features, we add an artificial weight μ to exclude such features from the optimal solution, avoiding randomness. The exact choice μ is not important as long as it is positive, which ensures that the respective feature will never be part of an optimal solution — see [92, Lemma 1.0]. The complete method is given as pseudocode in Algorithm 3.1.

Algorithm 3.1 QFS Algorithm: Binary search that, given an integer k, finds a value α^*, such that the optimal solution of a feature selection QUBO problem with matrix elements $Q_{ij}(\alpha^*, \epsilon, \mu)$, Eq. (3.19), contains k features. Source: [5].

```
**Input:**  $\boldsymbol{R}, \boldsymbol{I}, \epsilon, \mu, k$
  $a \leftarrow 0$
  $b \leftarrow 1$
  $\alpha \leftarrow 0.5$
  $\boldsymbol{z}^* \leftarrow Q^*(\alpha, \epsilon, \mu)$
  $k' \leftarrow \|\boldsymbol{z}^*\|_1$
  **while** $k' \neq k$ **do**
    **if** $k' > k$ **then**
      $b \leftarrow \alpha$
    **else**
      $a \leftarrow \alpha$
    **end if**
    $\alpha \leftarrow (a+b)/2$
    $\boldsymbol{z}^* \leftarrow Q^*(\alpha, \epsilon, \mu)$
    $k' \leftarrow \|\boldsymbol{z}^*\|_1$
  **end while**
  **return** $\alpha, \boldsymbol{z}^*$
```

3.3.1. Overview of Similar Approaches

There are several approaches that use similar techniques for performing FS, which shall be briefly compared to QFS. In [93], a feature ranking is computed by formulating FS as a quadratic programming (QP) task. To obtain the ranking, a real-valued weight vector is obtained by solving this proxy problem through dimension reduction. The QP task is only a relaxation of the corresponding QUBO problem with binary weight variables which we discuss in this chapter. As it is not guaranteed that the relaxation yields the exact QUBO solution [42], this approach is purely heuristic. In prospect, QUBO becomes tractable through quantum computation, which is why QFS does not need to resort to approximations or relaxations in order to become feasible.

As discussed earlier, resorting to a penalty term λ to restrict a solution to k features introduces several problems. Previous QUBO formulations based on redundancy and importance do exist [94], but rely on exactly this strategy. Choosing the value of λ is not trivial, and a very large value can drastically increase the dynamic range of the QUBO weight matrix (see Chapter 6). QFS weighs redundancy and importance against each other in order to fix k, rendering any additional constraints obsolete.

In [95], Grover's algorithm is applied to an oracle that yields the improvement in accuracy by adding or removing single features. This constitutes a quantum version of a simple sequential wrapper approach, improving the asymptotic runtime for greedily selecting the next feature in sequence, due to the theoretical property of Grover's algorithm to find a function's minimum or maximum with logarithmic time complexity. On the one hand, Grover's algorithm is not widely applicable to a large number of features in the NISQ era. On the other hand, the solution quality is only as good as the classical greedy algorithm. As Grover's algorithm is probabilistic in that it returns the correct minimum/maximum only with a certain probability, the result may be even worse than for the classical greedy approach.

3.4. Experimental Evaluation

So far, QFS has only been shown to work in theory. The following sections detail four different experiments to evaluate the performance of QFS. We use six evaluation data sets, both synthetic and taken from real-world data sources, which are listed in Table 3.1, and which shall be described briefly:

- `mnist` [96] contains 28×28 gray-scale images of handwritten digits. Each feature (i.e., each pixel) can take integer values from 0 (black) to 255 (white). We divided these values by 255 to rescale the features to the range $[0, 1]$. There are ten classes, one for each digit. This data set is challenging due to its high dimensionality.
- `ionosphere` [97, 98] contains measurements of electrons in the ionosphere captured by 16 antennae in north-east Canada. The resulting 34 features take values in the range $[-1, 1]$. The binary label indicates the presence of evidence of certain structures in the ionosphere.
- `waveform` is a synthetic data set first introduced in [99]. It contains short time series of length 21, each containing a random linear combination of two of three triangular base waves with added Gaussian noise. The $\binom{3}{2} = 3$ combinations of two out of three base waves provide the class label. There is considerable overlap between classes, making the correct classification challenging.
- `madelon` consists of 5-dimensional points sampled around the corners of a hypercube, with each corner randomly representing one of two classes. In addition, 15 linear combinations of these five features as well as 480 random irrelevant features ("probes") without predictive power are included, leading to 500 features in total.
- `synth_10` is another synthetic data set with $n = 10$ features and a binary label that indicates if a linear combination of a subset of four specific features is above a fixed threshold. We generated the data set by first randomly choosing $d_{\text{inf}} = 4$ indices of informative features $\mathcal{I} \subseteq \{1, \dots, n\}$ without replacement, with $n = 10$. We then sampled two random correlation matrices $\boldsymbol{C}_{\text{inf}}$ and $\boldsymbol{C}_{\text{rest}}$ with dimensions $d_{\text{inf}} \times d_{\text{inf}}$ and $d_{n-\text{inf}} \times d_{n-\text{inf}}$ respectively, using the algorithm in [100, Sec. 3.2] with parameter $\beta = 1$. Next, we sample i.i.d. $\mu_i \sim \mathcal{N}(0, 10)$ and $\sigma_i \sim \exp(\mathcal{N}(0, 1))$ for $i \in \{1, \dots, n\}$, such that $\boldsymbol{\mu}_{\text{inf}} = (\mu_i)_{i \in \mathcal{I}}$ and $\boldsymbol{\mu}_{\text{rest}} = (\mu_j)_{j \in \mathcal{I}^\complement}$, and $\boldsymbol{\sigma}_{\text{inf}}$ and $\boldsymbol{\sigma}_{\text{rest}}$ respectively (with $\mathcal{I}^\complement = \{1, \dots, n\} \setminus \mathcal{I}$). From this, we obtain covariance matrices $\boldsymbol{\Sigma}_{\text{inf}} = \boldsymbol{\sigma}_{\text{inf}} \boldsymbol{\sigma}_{\text{inf}}^\intercal \odot \boldsymbol{C}_{\text{inf}}$ and $\boldsymbol{\Sigma}_{\text{rest}}$ analogously. Finally, we sample a data point $\boldsymbol{x}$

Table 3.1.: Data sets used for numerical experiments: Number of features n, classes c, and number of samples S given. Source: [5].

Name	References	n	c	S
`mnist`	[96]	784	10	70 000
`ionosphere`	[97, 98]	34	2	351
`waveform`	[99, 98]	21	3	5000
`madelon`	[101, 98]	500	2	2000
`synth_10`	[5]	10	2	10 000
`synth_100`	[5]	100	2	10 000

with $\boldsymbol{x}_{\mathcal{I}} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{inf}}, \boldsymbol{\Sigma}_{\text{inf}})$ and $\boldsymbol{x}_{\mathcal{I}^\complement} \sim \mathcal{N}(\boldsymbol{\mu}_{\text{rest}}, \boldsymbol{\Sigma}_{\text{rest}})$. We generate the labels by sampling $\boldsymbol{w} \in \mathbb{R}^{d_{\text{inf}}}$ with $w_i \sim \mathcal{N}(0,1)$ i.i.d. for all $i \in \{1, \dots, d_{\text{inf}}\}$. Then, for each data point $\boldsymbol{x}$ we set the label to $y = 0$ if $\boldsymbol{w}^\intercal \boldsymbol{x} < \mathbb{E}_{\tilde{\boldsymbol{x}}}[\boldsymbol{w}^\intercal \tilde{\boldsymbol{x}}]$, and to $y = 1$ otherwise, which yields, in expectation, an equal class distribution.

- `synth_100` is generated using the same procedure as `synth_10`, but with $n = 100$ and $d_{\text{inf}} = 10$.

For the discretization step described in Section 3.1, we choose the number of bins $B = 20$ as a tradeoff between accuracy and computational speed. We further set $\epsilon = 10^{-8}$ and $\mu = \max_{i,j \in \{1,\dots,n\}} Q_{ij}(\alpha)$ in Algorithm 3.1. Both parameters were determined empirically by careful preliminary testing before performing the main experiments.

The first experiment in Section 3.4.1 serves as a proof of principle, in which we evaluate whether QFS can find any useful features at all. To this end, 30 features from the `mnist` data set are selected through QFS, and a range of 1-vs-all classifiers are trained on all digits. The performance of these classifiers is then compared to those trained on 30 random features, and on all available features, respectively, to assess the usefulness of the selected features. The second experiment in Section 3.4.2 constitutes a wider empirical comparison of various combinations of commonly used FS methods and ML models with the goal to show that QFS is competitive. In a third experiment in Section 3.4.3, QFS is applied in a more application-oriented setting, using the selected features as a means of lossy data compression. This is achieved by interpreting the reduced feature space as a latent representation, training a convolutional neural network to reconstruct the original features, and evaluating the quality of the reconstruction. Finally, we use quantum hardware in Section 3.4.4 to solve two exemplary feature selection Qubo instances for QFS, demonstrating that this method can indeed be used with currently available NISQ hardware.

3.4.1. Feature Subset Quality

The first experiment serves to verify that QFS is actually able to find an informative subset of features at all. For this purpose, we take the `mnist` data set and run Algorithm 3.1 ten

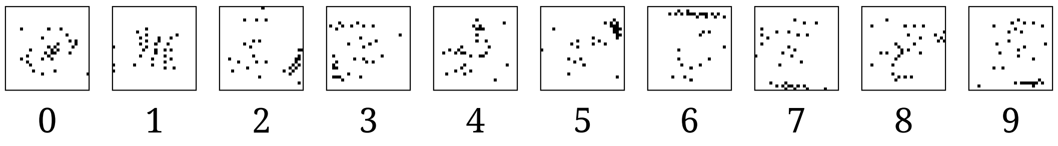


Figure 3.3.: Feature subsets found through QFS on all separate digits of the `mnist` data set. Black pixels represent selected features. Source: [5].

times with $k = 30$ such that we use about $3.8\,\%$ of the original 784 features (i.e., pixels). To solve the QUBO instances classically, we use the software package *qbsolv* [102]. For the redundancy measure, the pairwise MI matrix over all features is used, as specified in Eq. (3.9). As importance for digit d, the MI between the features x_i and a binary variable $\mathbb{1}\{y = d\}$ is calculated according to Eq. (3.8). This yields ten feature selections, one for each digit, which are used to perform a 1-vs-all classification. As a method to quantify whether the features selected by QFS are informative or not, a RF classifier is trained on these features, and its accuracy computed. For comparison, the accuracies of two RF classifiers that have been trained on *(i)* the whole feature set and *(ii)* a set of randomly selected features (uniformly sampled from the set of k-element subsets of $\{1, \dots, n\}$ for each digit) are reported, too. Each RF is composed of 100 estimators, each in turn a DT of maximal depth 5, using a maximum of 5 features when searching the best split. These restrictions can be justified by a need to limit the model size, which is a common objective in applications that use FS. For all classifiers, the Python implementation provided by *scikit-learn* [103] is used.

For the `mnist` data set, the selected features correspond to pixels. We show the pixels selected by QFS in Fig. 3.3. Concerning the RF models, we perform 10-fold cross validation and report mean and standard deviation of the classification accuracy, visualized in Fig. 3.4. For comparing randomly chosen feature selections, the cross-validated accuracy averaged over 5 random subsets is reported, resulting in mean and standard deviation over 50 runs in total.

The RF model consisting of the constrained estimators performs best on all digits using the optimal feature subset found through QFS, which confirms that this method indeed finds informative features that are useful for classification. Notably, the models trained using the QFS features outperforms not only the random subset models, but also the models that had access to all features. A restricted number of features per split in the base estimators leads to a higher chance of picking non-informative features, illustrating that FS as a pre-processing method can drastically improve model training.

3.4.2. Cross-Model Comparison

The second experiment serves to give a wider comparison of QFS to other feature selection methods. To this end, the accuracy of various classification models trained on the

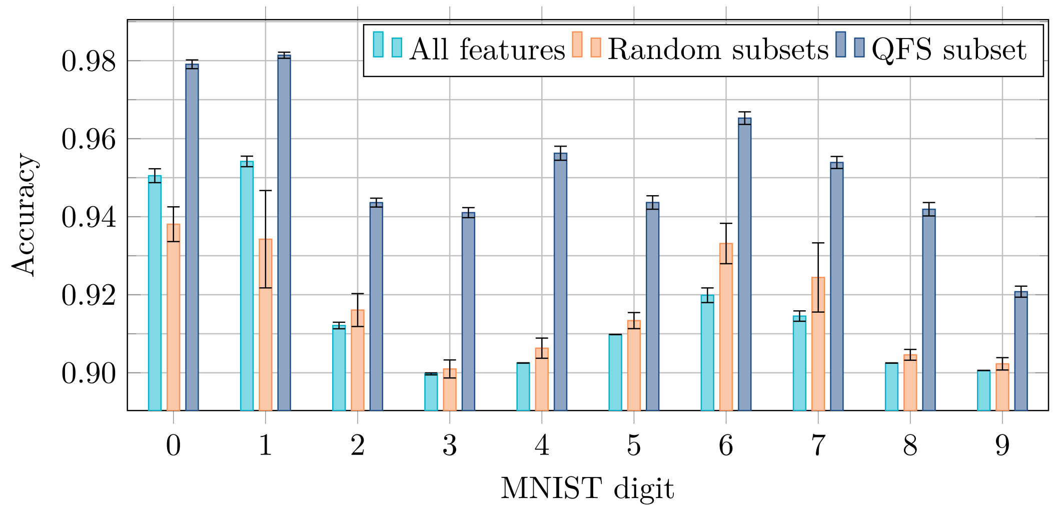


Figure 3.4.: Cross-validated accuracy of binary Random Forest classifiers of each separate digit of the `mnist` data set. One standard deviation is indicated. Source: [5].

respective feature subsets are compared in analogy to the previous experiment in Section 3.4.1.

This time, Algorithm 3.1 is applied to 5 different data sets to obtain feature subsets of fixed sizes k. For `madelon` and `synth_100` the known number of informative features is known, while for the remaining data sets we chose the feature subset sizes arbitrarily in varying percentage ranges of the total number of features. Thus, we arrive at 30 features (3.8 %) for `mnist`, 5 features (14.7 %) for `ionosphere`, 20 features (4 %) for `madelon`, 20 features (5 %) for `synth_100`, and 5 features (23.8 %) for `waveform`.

To evaluate the feature subset quality, we compare QFS to three other heuristic FS methods, namely 1. a feature ranking obtained from the Euclidean norm over the coefficients of n 1-vs-all Logistic Regression (LR) models, 2. a feature ranking obtained from the impurity-based feature importances given by an Extra Tree (ET) classifier with 100 estimators, and 3. Recursive Feature Elimination (RFE) [104] performed on a DT of maximal depth 10. The resulting subsets of features are then used to train 5 classification models, namely 1. an Artificial Neural Network with a single hidden layer of $\lfloor\sqrt{k}\rceil$ neurons and Rectified Linear Unit (ReLU) activation, 2. 1-vs-all Logistic Regression, 3. a single DT, 4. a RF with 100 DTs, and 5. a Naive Bayes classifier with Gaussian prior. For better comparability, the number of hidden layers in the ANN is chosen such that dependency between the number of parameters and selected features k is linear. Both the ANN and the LR have a fixed budget of 1000 learning iterations. Again, the Python implementations provided by *scikit-learn* [103] are used for all models and FS methods. On every model a 10-fold cross validation is performed, and mean and standard deviation of the classification accuracy is reported.

The results are visualized in Fig. 3.5 and show that QFS compares generally favorably among FS methods: QFS is mostly within the standard deviation of the top-performing FS methods. While other methods seem to perform worse on specific data sets (e.g., LR on `mnist`, and RFE on `ionosphere`), QFS is consistently among the best algorithms, a notable outlier being the ANN classifier on `waveform`, which did not properly converge on the selected feature subset. The RFE method using decision trees often leads to better accuracies, but comes at the cost of much higher computation time, owing to the fact that it is a wrapper method which requires the model to be re-trained in every iteration, which is very resource-inefficient for complex models.

For `madelon` and `synth_100` the ground-truth informative features are known, which can be exploited to evaluate the distance between the truly optimal feature subset and the subset found by each FS method. As a distance measure, we use the edit distance between pairs of feature subsets, which is the number of features that need to be swapped in order to turn one subset into the other – this is equivalent to the Hamming distance between the binary feature indicator vectors, divided by two. The subsets generated by each FS method used in this experiment are represented by nodes in an undirected graph, the edge weights giving the pairwise edit distances. If two or more nodes have distance 0, they are represented as clusters. The resulting graphs are shown in Fig. 3.6.

QFS is able to find all informative features for `synth_100`, and is closest to ground-truth on `madelon` among all other methods except for the ET classifier ranking, which is able to find the ground-truth features in both cases. In this experiment, the ET classifier ranking is particularly effective, producing the optimal feature subsets on both data set. However, as a wrapper method, it comes at a much higher computational cost than the other methods. The LR ranking is furthest from ground truth, and also furthest from all other methods, suggesting that LR is a too simple model for these data sets. RFE is behind the performance of both QFS and ET, being unable to find ground truth in both cases. This result demonstrates that MI is a useful measure of redundancy and importance in QFS, as it produces feature subsets close or identical to ground truth.

3.4.3. Lossy Compression with an Autoencoder

A different perspective on the selection of important features is the removal of *unimportant* features. This implies that QFS can be used as a type of lossy compression by computing an optimal feature subset S and discarding all features $i \notin S$. Feeding this compressed representation into a suitable ML model, we can attempt to reconstruct the original features [105], as shown schematically in Fig. 3.7. This third experiment evaluates this idea of lossy compression empirically.

For our experimental setup, we perform QFS on `mnist` with $k = 25$ (i.e., $3.19\,\%$ of all image pixels). The resulting subset S of pixel positions can be interpreted as a type of latent space, i.e., a compressed data representation similar to representations obtained from Principal Component Analysis (PCA) [9, Eq. 3.48] or an autoencoder [106].

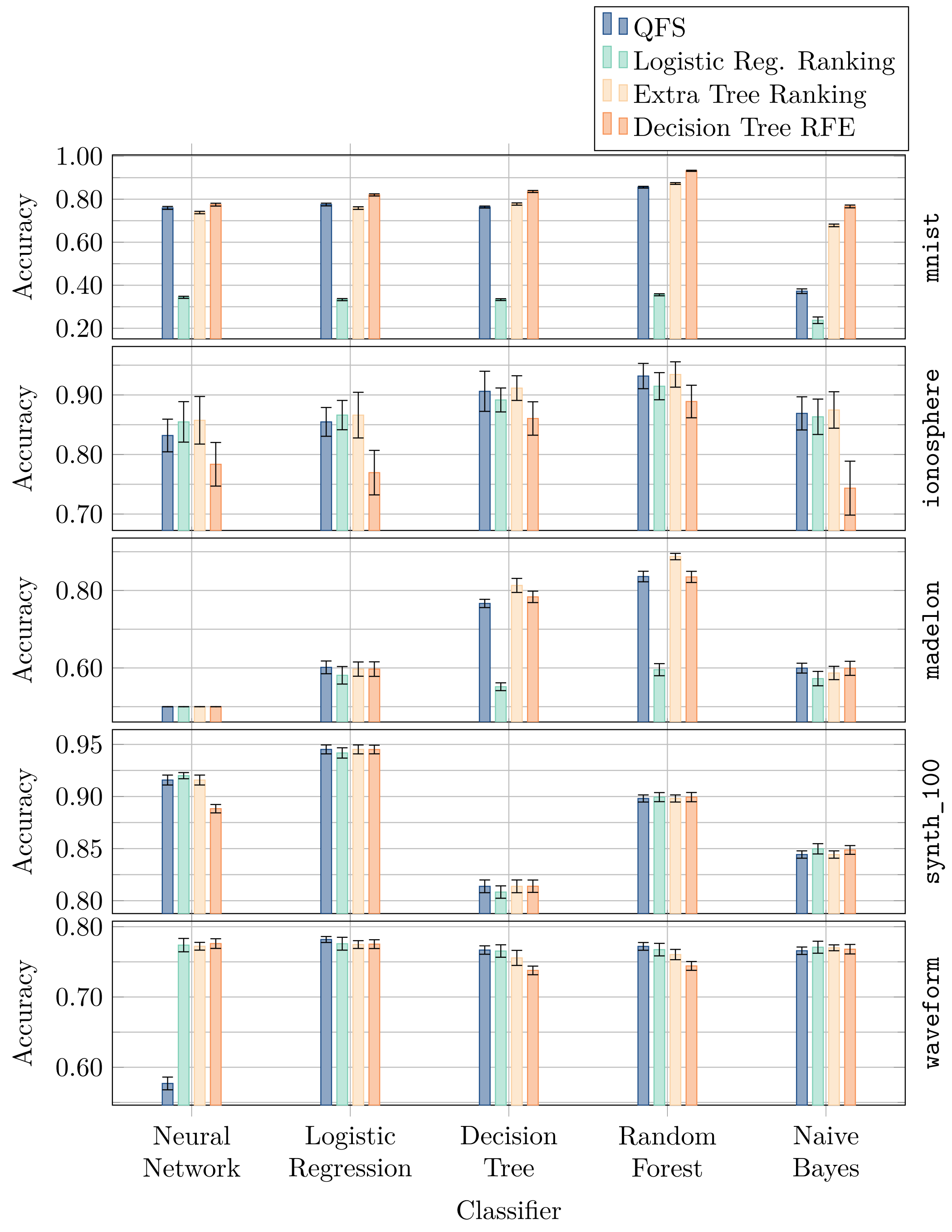


Figure 3.5.: Cross-validated accuracy of various classifiers using feature subsets produced by various feature selection methods on three different data sets. One standard deviation is indicated. Source: [5].

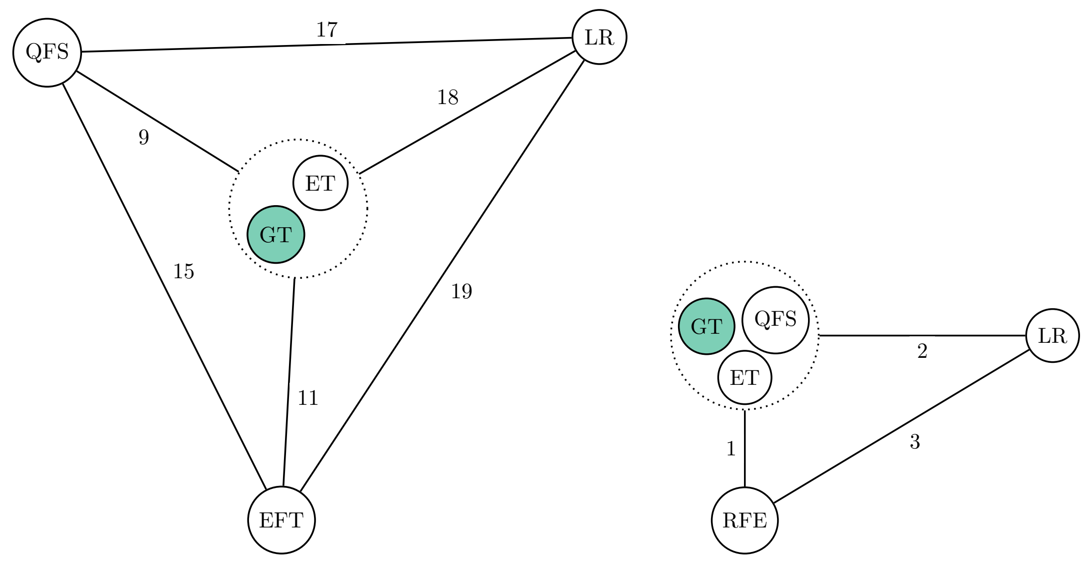


Figure 3.6.: Edit distances between feature subsets found through the different FS methods on data sets `madelon` and `synth_100`, for which the ground truth informative features are known. Nodes represent FS methods, edge weights give the edit distance between feature subsets. Ground truth (GT), LR ranking (LR), ET classifier ranking (ET), RFE and QFS are shown. Methods within dotted circles have distance 0. Source: [5].

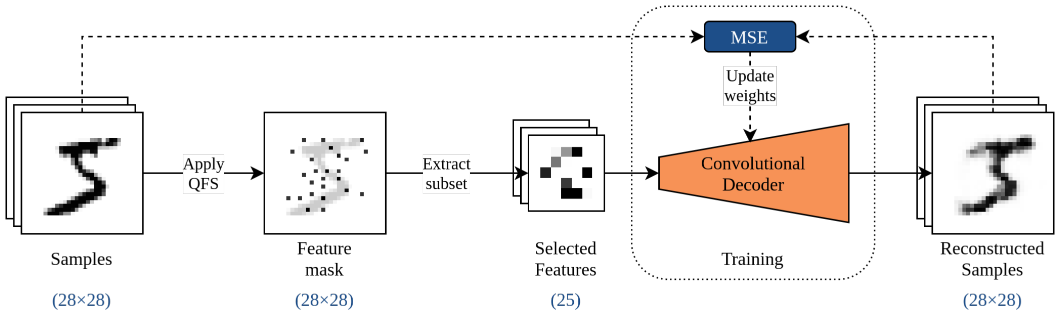


Figure 3.7.: Experiment 3: Compression and decompression pipeline using QFS with $k = 25$ and a convolutional decoder. Source: [5].

The resulting compressed representation of the digits is fed into a Convolutional Neural Network (CNN) that projects the images back to a size of 28×28 while minimizing the difference between reconstruction and original as a loss function. Accordingly, the CNN architecture consists of a linear input layer with k inputs and 392 outputs. The output is reshaped to 8 channels of size 7×7. Two sequential 2D transposed convolution operations interspersed with ReLU activations serve to first inflat the images to $16 \times 14 \times 14$ as an intermediate representation, and finally to $1 \times 28 \times 28$, which is the original image size. A sigmoid function is applied to the output in order to ensure that pixel values are

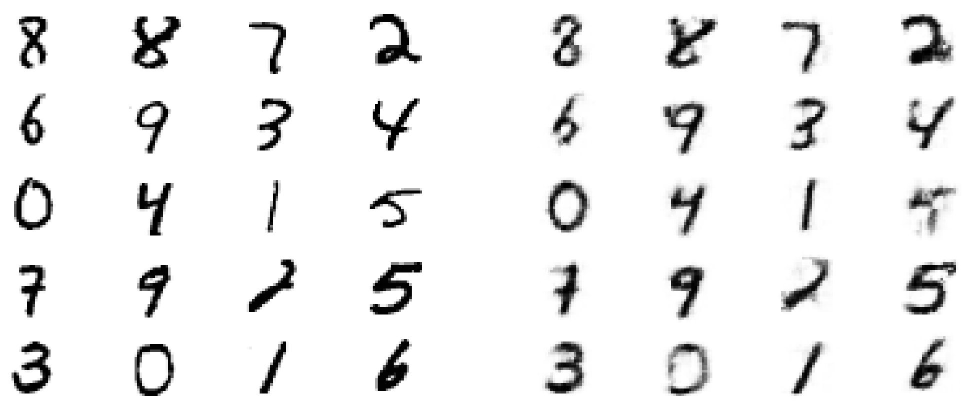

(a) Original samples **(b)** Reconstruction

Figure 3.8.: Visual comparison of `mnist` samples reconstructed from 25 pixels at fixed positions selected through QFS. Source: [5].

between 0 and 1. As the loss function, the MSE between the original samples $\boldsymbol{x}$ and the reconstructions is used:

$$\frac{1}{784}\|\boldsymbol{x} - f_{\boldsymbol{\theta}}(\boldsymbol{x}_S)\|_2^2 \longrightarrow \min_{\boldsymbol{\theta}}, \tag{3.20}$$

where $f_{\boldsymbol{\theta}}$ is the model function with weights $\boldsymbol{\theta}$, and $\boldsymbol{x}_S$ the sub-vector of $\boldsymbol{x}$ containing only the features in S found through QFS. The model is trained for 1000 epochs with batches of size 250, using the Adam optimizer [107] from *PyTorch* [108] with a 1cycle learning rate scheduler [109] using a maximum learning rate of 0.01.

After training, the CNN achieves a MSE of 21.7389, corresponding to an average squared deviation per pixel of 0.0277. As a visual reference, Fig. 3.8 shows 20 random `mnist` samples on the left, and their respective reconstructions on the right. The reconstructed samples are visually very similar to the originals, suggesting that the features (i.e., the pixel positions) found by QFS indeed contain useful information about the samples that help the CNN model to infer the values of neighboring pixels in the image.

3.4.4. QFS on Quantum Hardware

Up to this point, only classical Qubo solvers have been used for performing QFS. In this fourth experiment, QFS is applied using actual quantum hardware as the Qubo oracle in Algorithm 3.1. To this end, we use both a quantum annealer as well as a quantum gate computer to test the effectiveness of both major paradigms of QC for this task.

We construct the QFS Qubo instances for three data sets, `ionosphere`, `waveform`, and `synth_10` as before, but solve them using QA on a D-Wave quantum annealer and VQE on an IBM gate quantum computer (see Section 2.4.2). In preparation for this experiment, first an entirely classical run of Algorithm 3.1 is performed to obtain values for α for a predefined number of selected features k. While in principle, this search for α can be done

Table 3.2.: Number of features n, selected features k, and resulting value of α given by Algorithm 3.1. Source: [5].

data set	n	k	α
`ionosphere`	34	5	0.906 25
`waveform`	21	5	0.781 25
`synth_10`	10	4	0.875 00

on quantum hardware as well, pre-computing α classically reduces computation time significantly without changing the experiment's result. Saving computation time on NISQ quantum gate hardware is critical, as IBM imposes a hard time limit on their devices. For the same reason, only the `synth_10` data set for the VQE algorithm is considered.

The determined α values are listed in Table 3.2 along with the chosen number of selected features k. On this basis, QUBO instances for each data set are computed and solved on both hardware platforms.

The D-Wave quantum annealer *Advantage 5.1* operating on 5627 qubits was accessed via D-Wave's cloud service *Leap*[11]. To perform QA, the implementation provided by *ocean*[12] with the *DWaveSampler* and default parameters was used to evaluate all three QUBO instances for the respective data sets. A total number of 1024 samples were obtained per QUBO instance, each representing an estimate of the solution. Additionally, the same experiment was repeated using Simulated Annealing, again using D-Wave's Python implementation contained in *ocean* with default parameters.

The IBM quantum gate computer *ibmq_ehningen*, on the other hand, operates on a *Falcon r5.11* processor with 27 qubits[13] and is accessible via IBM's cloud service *IBM Quantum*[14]. To perform the experiment, Qiskit Runtime's default VQE implementation [79] with the Simultaneous Perturbation Stochastic Approximation (SPSA) optimizer [110] was run for 32 iterations using Qiskit's default parameters. A Pauli Two-Design [111] with four layers was used as the parametric ansatz. As mentioned earlier, computations on the quantum gate hardware is time-consuming and, at the same time, strictly limited. For this reason, only the QUBO instances for `synth_10` were evaluated, again taking 1024 samples from the resulting quantum circuit.

QA results are shown in Fig. 3.9, which shows all 1024 samples sorted in ascending order of energy, such that the lowest measured energies are on the left, and mean and standard deviation is reported over the sorted sequences of 16 runs. For better comparison, the globally optimal energies were found by brute force, and are shown as horizontal lines. On the right, the number of times each solution bit was measured as 1 across all shots

[11]`https://cloud.dwavesys.com/leap` (last accessed June 3, 2025)

[12]`https://docs.ocean.dwavesys.com/en/stable/index.html` (last accessed June 3, 2025)

[13]`https://www.fraunhofer.de/content/dam/zv/de/institute-einrichtungen/Kooperationen/kompetenznetzwerk-quantencomputing/brochure_fraunhofer-v10.pdf` (last accessed June 3, 2025)

[14]`https://quantum-computing.ibm.com`

is shown. As usual, the bits correspond to feature indices for QFS. Again, mean and standard deviation over 16 runs is reported. The color of each bar indicates whether the corresponding global optimum of the respective bit is 0 or 1. The sequence of optimal bits corresponds to the optimal feature selection for our application. Fig. 3.10 is analogous to Fig. 3.9, showing the results obtained through Simulated Annealing.

The histograms show a clear correspondence between feature optimality (bar color) and the number of occurrences, which indicates that QA is able to find the global optimum in a certain fraction of samples. As the number of qubits increases, this correlation gets less and less pronounced. The optimum was found in $10.78 \pm 5.12\%$ for `synth_10`, $0.18 \pm 0.18\%$ for `waveform`, and only a single time across all 16 runs for `ionosphere`. Simulated Annealing, in contrast, finds the correct bits with higher probability, even for data of higher dimension: The optima are found in $100.00 \pm 0.00\%$ of shots for `synth_10`, $20.39 \pm 1.40\%$ for `waveform` and $21.04 \pm 1.02\%$ for `ionosphere`. From this can be deduced that the use of NISQ hardware compromises the solution quality. This may be due to a a number of causes, such as loss of precision when loading the Qubo weight matrix onto the quantum annealer, or noisy read-out from the quantum device.

The single result for VQE is shown in Fig. 3.11, analogous to Fig. 3.9, with mean and standard deviation obtained from 5 runs. In contrast to QA, this result is much less promising, showing no immediate correspondence between the optimal bits and number of occurrences. Only in $0.14 \pm 0.08\%$ of measurements was the global optimum found. This leads to the assumption that hardware noise, as well as the low number of optimization steps that were used due to long run times, had a detrimental impact on the performance. Possibly VQE could perform better for longer run times and less noisy hardware. Importantly, the VQE results are different from the D-Wave results in that the 1024 VQE samples were all taken from one optimized circuit, while each shot represents one approximation run to find the underlying QFS optimum on the QA device.

All in all we conclude from this experiment that near-term quantum devices can in principle be used for QFS today. However, best performance is to be expected on low-dimensions data using special-purpose hardware tailored for Qubo, like quantum annealers.

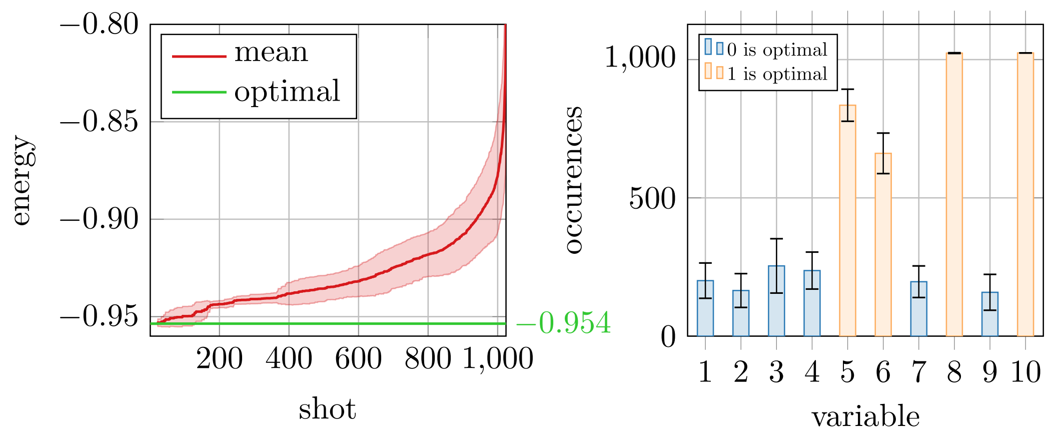
3.5. Concluding Remarks

This chapted has described the QFS algorithm for performing feature selection based on a Qubo embedding, which can be solved on both classical and quantum hardware. It uses mutual information as a basis for measures of importance and redundancy, which are balanced using an interpolation factor α. It was shown that QFS allows the selection of feature subsets of any desired size k, as was proven theoretically, without additional constraints on the solution space, which could negatively impact the solvability of the Qubo instances. In a range of experiments the framework's effectiveness was demonstrated by comparing different common features selection methods and the resulting performance on different ML models, as well as a practical application for lossy data compression.

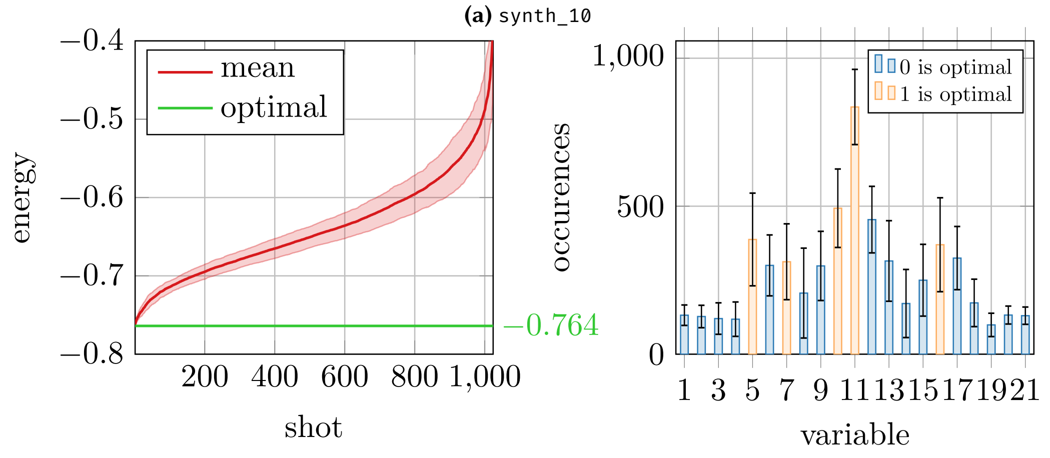
Another experiment was successfully performed on actual quantum hardware, further demonstrating that QFS is NISQ-compatible, at least on low-dimensional data and using QA. The feasability is dictated by available hardware, but it can be expected that QFS scales in accordance with future quantum computing developments.

The choice of MI as the basis for measures of importance and redundancy is flexible: As Proposition 3.1 is valid for general $\boldsymbol{I}$ and $\boldsymbol{R}$ with non-negative entries, the proof holds for any combination of importance and redundancy measures, and Algorithm 3.1 can be applied accordingly. Possible alternative choices include entropy, Pearson correlation, or other information-theoretic measures. Potentially, expert knowledge, if available, can be incorporated by assigning manual, application-specific importance weights, or forbidding certain pairs of features through manual penalty weights.

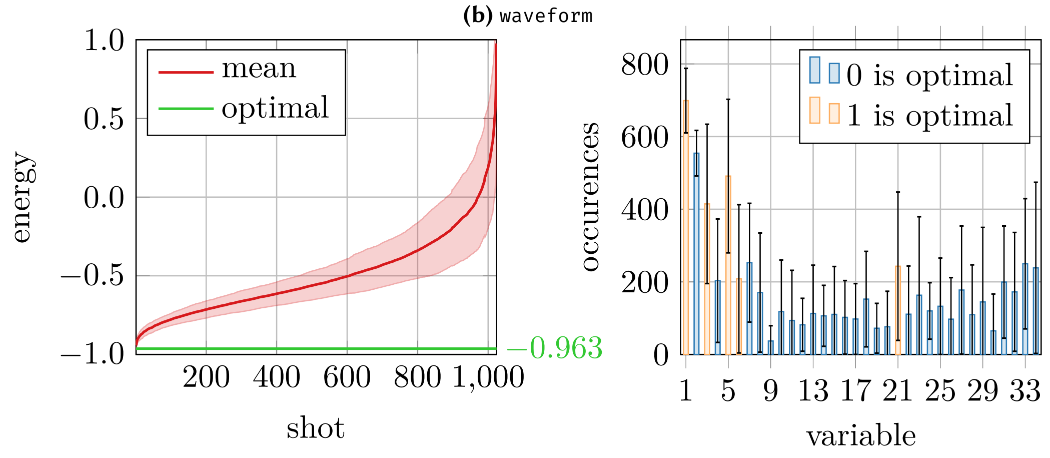
FS serves as the first example of an application of QC – more precisely, Quantum Annealing – to a problem arising in ML. In Part IV we will see several other examples of QA solving interesting and non-trivial optimization problems. A central advantage of QA is its potential to optimize over the entire search space at once, converging to the globally optimal solution, given the annealing process is carried out sufficiently slowly and the noise caused by imperfect hardware does not skew the energy landscape too much. In Chapter 6 we investigate the last point thoroughly and propose methods to avoid such errors.



(a) `synth_10`

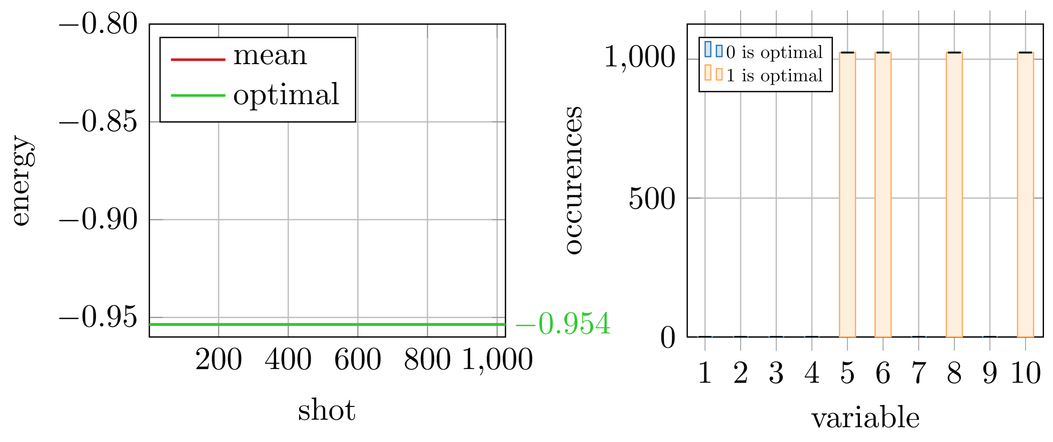


(b) `waveform`

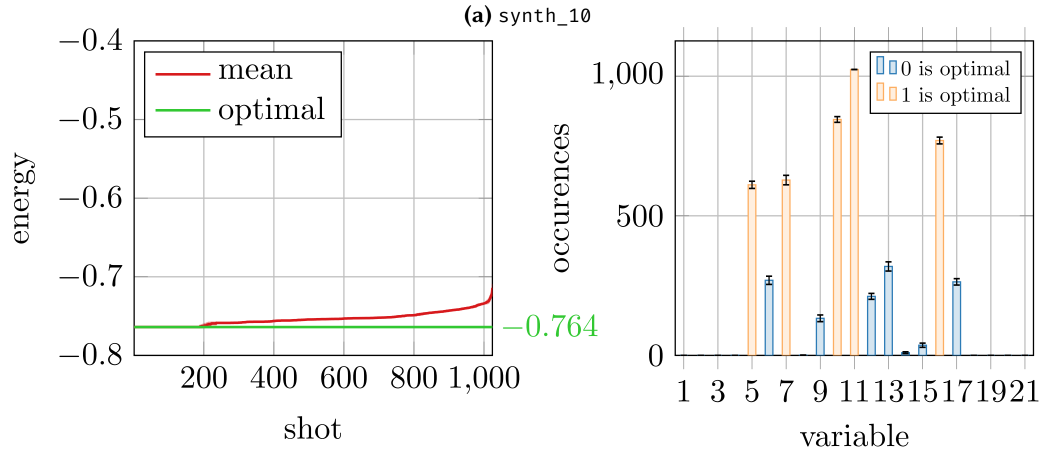


(c) `ionosphere`

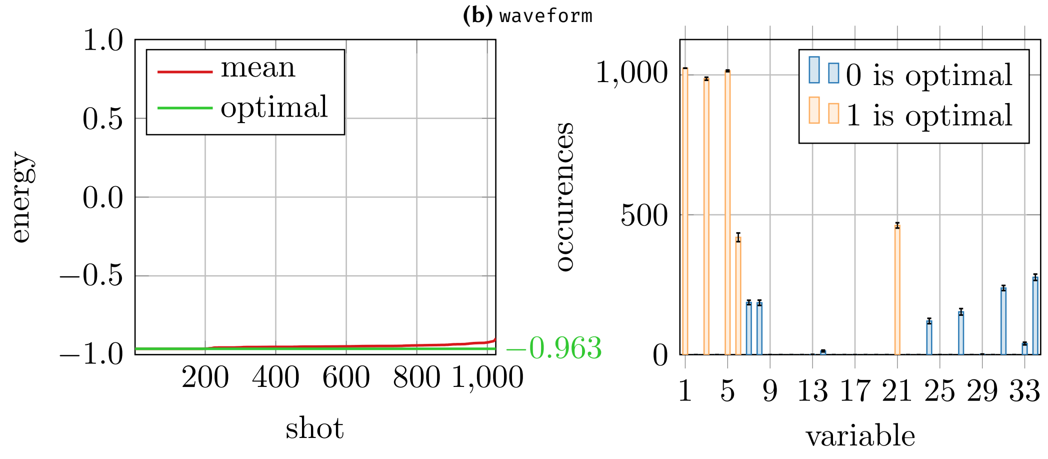
Figure 3.9.: Histograms of samples from the D-Wave quantum annealer performing QFS on three different data sets. **Left:** Shots sorted by energy (mean and standard deviation over 16 runs), minimal energy shown as horizontal lines. **Right:** Prevalence of solution bits, optimal bits shown in yellow. Mean and standard deviation over 16 runs is reported. Source: [5].



(a) `synth_10`



(b) `waveform`



(c) `ionosphere`

Figure 3.10.: Same results as in Fig. 3.9, but using SA instead of QA. Source: [5].

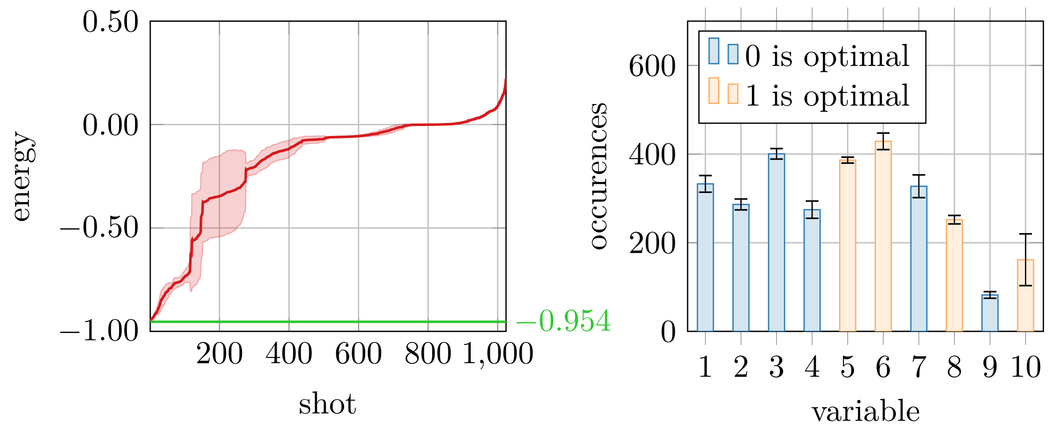


Figure 3.11.: Samples from the IBM quantum gate computer performing QFS on `synth_10` in analogy to Fig. 3.9. Minimal energy shown as horizontal line, optimal bits shown in yellow. Source: [5].

4. Quantum Support Vector Machines

As we have seen in Section 2.3.2, the SVM is (in its original form) a classification model building on the idea of separating a data set containing points from two classes with a hyperplane, such that all points of class -1 are on one side, and all points of class $+1$ are on the other. In addition, the hyperplane is chosen such that the distance to the nearest points is maximized, hoping to achieve best possible generalization.

SVMs can be trained by solving the optimization problem in Def. 2.11, e.g., using quadratic programming for its primal or dual formulation [9]. The resulting parameters $\boldsymbol{w}$ and b are then used to compute the prediction of new data points. In this chapter we take a different route by considering different strategies to employ QC. As discussed in Section 2.4, QC with its two paradigms of AQC and GQC can be used as a tool for both optimization and computation. Therefore, we will apply both to different aspects of the SVMs, namely training and deployment.

In Section 4.1 we formulate the problem of SVM training as a Qubo problem, which allows us to find the set of support vectors from a data set of labeled points using a quantum annealer. To this end, we need to tweak the optimization problem in Def. 2.12 to be compatible with binary optimization. In Section 4.2 we use GQC to embed a trained SVM model with parameters $\boldsymbol{w}$ and b onto a quantum circuit, which allows us to compute predictions given new data points. Notably, if this circuit is evaluated repeatedly, the prediction converges to the classical model's output. This is valuable, as the SVM is a well-understood model with many favorable theoretical properties, which the quantum version preserves. Taken together, this chapter provides the building blocks for a quantum classification pipeline, which constitutes a step in the direction of quantum-ready ML.

This chapter is based on publications [7, 2]. Section 4.1 uses the SVM formulation from [7], which the author of this this thesis developed. Section 4.2 is based on [2], for which the author jointly developed the RQSVM model, implemented an efficient simulator, and conducted and documented experiments.

4.1. SVM Training through Quantum Annealing

Recall Def. 2.12 of the dual SVM optimization problem given in Section 2.3.2, which we shall re-state here for simplicity:

Definition 2.12 (Dual SVM [9]). *Let $\mathcal{D}$ as in Def. 2.11, with $\boldsymbol{y} \in \mathbb{R}^N$ the vector containing all y_i in $\mathcal{D}$, $\boldsymbol{X}$ the $N \times d$ data matrix as defined in Def. 2.9, and $\boldsymbol{K} = \varphi(\boldsymbol{X})^\intercal \varphi(\boldsymbol{X})$ the* kernel matrix *w.r.t. some feature map φ. The dual form of the SVM training problem given in Def. 2.11 is*

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\mathit{Maximize}}\ & \mathbf{1}_N^\intercal \boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^\intercal(\boldsymbol{y}\boldsymbol{y}^\intercal \odot \boldsymbol{K})\boldsymbol{\alpha} \\ \mathit{s.t.}\ & 0 \leq \alpha_i \leq C\ \forall\, i \in \{1,\dots,N\}, \\ & \boldsymbol{\alpha}^\intercal \boldsymbol{y} = 0. \end{aligned}$$

For simplicity, we focus on the linear SVM by fixing the feature map φ to the identity function, leading to $\boldsymbol{K} = \boldsymbol{X}^\intercal \boldsymbol{X}$ being the Gram matrix. If we take a closer look at the equation, we find that it has strong similarity with a Qubo problem, although with a few differences, namely

- we have a maximization instead of a minimization problem,
- the values $\boldsymbol{\alpha}$ that we optimize are real-valued, not binary, and
- there are additional constraints on α.

Firstly, we can flip the sign to convert a maximization to a minimization problem. The second point, however, is more challenging, because quantum annealers can only optimize over binary variables $\boldsymbol{z} \in \mathbb{B}^N$. To resolve this, we can take a radical approach and binarize the elements of $\boldsymbol{\alpha}$ and simply set $\alpha_i = Cz_i$, meaning that α_i can take *either* 0 or C instead of being a real value in the interval $[0, C]$. By doing this, we automatically inforce the first set of constraints, namely $0 \leq \alpha_i \leq C$. The second constraint, $\boldsymbol{\alpha}^\intercal \boldsymbol{y} = 0$, can be inforced by adding a penalty term

$$\lambda(\boldsymbol{\alpha}^\intercal \boldsymbol{y})^2 = \lambda \cdot \boldsymbol{\alpha}^\intercal \boldsymbol{y}\boldsymbol{y}^\intercal \boldsymbol{\alpha}, \tag{4.1}$$

which is 0 when the condition is fulfilled, and assumes a positive value otherwise. We have to choose a value $\lambda > 0$ that is large enough to make any solution that violates the constraint non-optimal (see Section 2.2).

Combining these ideas, we arrive at the following Qubo formulation of the binarized SVM learning problem:

Definition 4.1 (Binary Qubo-SVM [7]). *Let $C, \lambda > 0$, and let $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ and $\boldsymbol{y} \in \mathbb{S}^N$. The Binary Qubo-SVM can be trained by solving the Qubo problem given by*

$$f_{\text{SVM}}(\boldsymbol{z}; C, \lambda, \boldsymbol{X}, \boldsymbol{y}) = -\mathbf{1}_N^\intercal \boldsymbol{z} + C\boldsymbol{z}^\intercal \left(\frac{1}{2}(\boldsymbol{y}\boldsymbol{y}^\intercal \odot \boldsymbol{K}) + \lambda \boldsymbol{y}\boldsymbol{y}^\intercal\right)\boldsymbol{z}.$$

Given a minimizer $\boldsymbol{z}^* \in \mathbb{B}^N$, the parameters $\boldsymbol{w}$ and b used for the prediction function (see Eq. (2.11)*) are then given by*

$$\boldsymbol{w} = C \cdot (\boldsymbol{z}^* \odot \boldsymbol{y})^\intercal \boldsymbol{X} = C \sum_{i=1}^{N} z_i^* y_i \boldsymbol{x}^{i\intercal}$$
$$b = \boldsymbol{w}^\intercal \boldsymbol{x}^{i\intercal} - y_i \text{ for any } i \in \{1, \dots, N\},$$

where $\boldsymbol{x}^i$ is the i-th row of the data matrix, i.e., the i-th data point as a row vector.

For greater numerical stability, b should be computed over several (or all) indices i and the results averaged. Conveniently, the solution vector $\boldsymbol{z}^*$ to the Qubo problem is at the same time an indicator vector of which data points are support vectors, i.e., which vectors contribute to the decision boundary.

4.1.1. Increasing Precision

Since the initial publication of the binarized SVM [7], a few authors have used the same technique independently and extended it to allow for greater precision by means of representing each α_i with more than one qubit [112, 39]. For completeness, we elaborate on these ideas and include them in our experimental evaluation.

The simplifying assumption of $\alpha_i = Cz_i$ is quite radical in that it allows no gradation between being no support vector at all ($\alpha_i = 0$) or being misclassified ($\alpha_i = C$). From another point of view, data points can either not contribute to the optimal $\boldsymbol{w}$ at all, or equally with weight C.

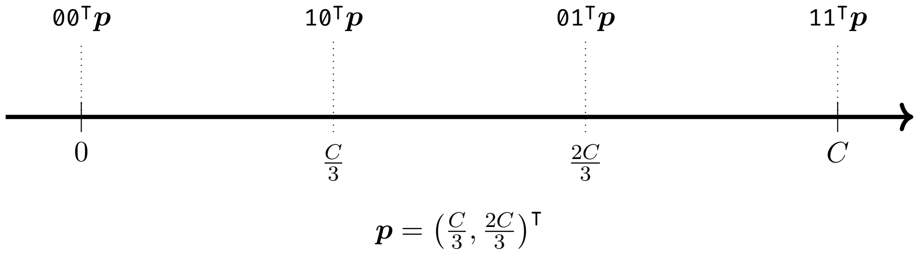
We can soften this restriction by encoding the weights α_i using $k > 1$ bits instead of only one. Assume we have a vector $\boldsymbol{p} = (p_1, \dots, p_k)^\intercal \in \mathbb{R}_{0+}$ with $\|\boldsymbol{p}\|_1 = C$. Then we can represent each α_i as a sum $\sum_{j=1}^{k} p_j z_{i,j}$ with $z_{i,j} \in \mathbb{B}\ \forall j \in \{1, \dots, k\}$, such that for each combination of bits that sum is bounded between 0 and C. To do this for all α_i, let now $\boldsymbol{z} = (z_{1,1}, \dots, z_{1,k}, z_{2,1}, \dots, z_{N,k})^\intercal \in \mathbb{B}^{Nk}$ be a binary vector where the k bits for each α_i are simply concatenated. We can construct a matrix that maps this k-times larger vector to an N-element vector utilizing the Kronecker product:

$$\boldsymbol{P} = \boldsymbol{I}_N \otimes \boldsymbol{p}^\intercal \in \mathbb{R}^{N \times Nk}. \tag{4.2}$$

This way, we obtain $\boldsymbol{\alpha} = \boldsymbol{P}\boldsymbol{z} \in [0, C]^N$ for any $\boldsymbol{p} \in \mathbb{R}_{0+}^k$ with $\|\boldsymbol{p}\|_1 = C$. Using $\boldsymbol{P}\boldsymbol{x}$ as a replacement for $\boldsymbol{\alpha}$ in Def. 4.1 yields the following definition.

Definition 4.2 (k-bit Qubo-SVM)**.** *Let $C, \lambda > 0$, $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ and $\boldsymbol{y} \in \mathbb{S}^N$ as before. Further, let $\boldsymbol{p} \in \mathbb{R}^k$ with $\|\boldsymbol{p}\|_1 = C$. The k-bit Qubo-SVM can be trained by solving the Qubo problem given by*

$$f_{\text{SVM},\boldsymbol{k}}(\boldsymbol{z}; C, \lambda, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{p}) = -\boldsymbol{1}_N^\intercal \boldsymbol{P}\boldsymbol{z} + \boldsymbol{z}^\intercal \boldsymbol{P}^\intercal \left(\frac{1}{2} (\boldsymbol{y}\boldsymbol{y}^\intercal \odot \boldsymbol{K}) + \lambda \boldsymbol{y}\boldsymbol{y}^\intercal \right) \boldsymbol{P}\boldsymbol{z}, \tag{4.3}$$



$00^\intercal \boldsymbol{p}$
$10^\intercal \boldsymbol{p}$
$01^\intercal \boldsymbol{p}$
$11^\intercal \boldsymbol{p}$
0
$\frac{C}{3}$
$\frac{2C}{3}$
C
$\boldsymbol{p} = \left(\frac{C}{3}, \frac{2C}{3}\right)^\intercal$

Figure 4.1.: Example of a conversion from a binary vector of length $k = 2$ to a scalar value between 0 and C through multiplication with a vector $\boldsymbol{p}$; by choosing $p_j = C \cdot 2^{j-1}/(2^k - 1)$, the interval $[0, C]$ is sampled evenly.

where $\boldsymbol{P} = \boldsymbol{I}_N \otimes \boldsymbol{p}$. Given a minimizer $\boldsymbol{z}^* \in \mathbb{B}^{Nk}$, the parameters $\boldsymbol{w}$ and b used for the prediction function (see Eq. (2.11)*) are then given by*

$$\boldsymbol{w} = ((\boldsymbol{P}\boldsymbol{z}^*) \odot \boldsymbol{y})^\intercal \boldsymbol{X}$$
$$b = \boldsymbol{w}^\intercal \boldsymbol{x}^{i\intercal} - y_i \text{ for any } i \in \{1, \dots, N\},$$

where $\boldsymbol{x}^i$ is the i-th row of the data matrix, i.e., the i-th data point as a row vector.

The question remains how to effectively choose $\boldsymbol{p}$. As the encoding of the α_i values already resembles the base-2 number system, it is natural to choose powers of 2 to subdivide the space $[0, C]$ evenly, in a way that $\boldsymbol{0}_k$ represents 0, and $\boldsymbol{1}_k$ represents C. To achieve this, let $p_j = C \cdot 2^{j-1}/(2^k - 1)$ for all $j \in \{1, \dots, k\}$. It is easy to see that

$$\sum_{j=1}^{k} \frac{C \cdot 2^{j-1}}{2^k - 1} = \frac{C}{2^k - 1} \sum_{j=0}^{k-1} 2^j = C,$$

and the interval $[0, C]$ is subdivided evenly, as visualized in Fig. 4.1. Another advantage of choosing this $\boldsymbol{p}$ is that the special case $k = 1$ yields the original Binary Qubo-SVM given in Def. 4.1. Therefore, we can focus on Def. 4.2 in the following evaluation.

4.1.2. Experimental Evaluation

It remains to be investigated how the discretization of $\boldsymbol{\alpha}$ affects the quality of the trained classifier. To test this, we train the k-bit Qubo-SVM on a range of data sets for various values of C and k and record their test accuracies. For comparison, we also train a classical linear SVM using LIBSVM [113], which is contained as part of the scikit-learn Python package [103].

The data sets we use are listed in Table 4.1. All three data sets are widely used as classification benchmarks throughout literature, two of which we have already seen in Section 3.4. As the SVM (in its original form) can only separate two classes, we modify both `iris` and `mnist`: For iris, we only use the classes *versicolor* and *virginica*, which leaves $N = 100$

Table 4.1.: Data sets used for numerical experiments. For each data set, the number of features d and data points N is given.

Name	References	d	N
`iris` (mod.)	[114]	4	150
`sonar`	[115]	60	208
`mnist` (mod.)	[96]	196	200

data points. For `mnist`, we only use 100 samples each of the digits 4 and 7 as our two classes. To reduce the number of features, we perform max-pooling over every 2×2 pixel block, reducing the image size to 14×14 pixels. We normalize the values to the interval $[0, 1]$ and then linearize the images to vectors of length 196, arriving at a data set with $N = 200$ and $d = 196$.

To get a wider overview over the model's performance, we test all combinations of values $C \in \{2^{-6}, 2^{-5}, \dots, 2^{4}\}$ and $k \in \{1, 2, 3\}$. For choosing λ we use an iterative approach where we start with $\lambda = 1$, solve the QUBO problem and check if the constraint $\boldsymbol{z}^{*\intercal}\boldsymbol{P}^{\intercal}\boldsymbol{y} = 0$ is violated. If it is, we double λ and try again until we find a valid solution.

As a solver, we use the MST2 multistart tabu search algorithm [116] contained in the dwave-tabu Python package[15], which performs one million restarts per run. To further increase the solution quality, we perform 20 runs for each QUBO instance. In addition, we tried to solve the QUBO problems on a D-Wave quantum annealer. However, we found that Eq. (4.3) produces dense weight matrices for which the system was not able to find an embedding onto its qubit topology.

We performed a 5-fold cross validation and report the mean prediction accuracy in Fig. 4.2, making sure to use the same splits across all models and hyperparameter settings for comparability. Remarkably, despite the strong simplification of using even just two discrete values for all α_i, the accuracy scores are comparable to the LIBSVM results that use floating-point values. For small C, the QUBO-SVM even surpasses LIBSVM on `iris` and `sonar`. Surprisingly, increasing the precision by choosing a larger k does not generally lead to higher accuracy; only on `mnist` a higher k-value leads to better solutions for larger C. This implies that using a higher number of support vectors that contribute equally to the parameter vector $\boldsymbol{w}$ is more useful than allowing for more fine-granular weighting of fewer support vectors, which is an interesting insight, whose generality needs to be investigated more thoroughly in future work. However, we observe that from a certain C value we cannot find suitable solutions on both `iris` and `mnist`, as the minimizing vector is $\mathbf{0}_{Nk}$, and $\boldsymbol{\alpha} = \mathbf{0}_N$ accordingly. Without any support vectors we cannot correctly compute $\boldsymbol{w}$, and the resulting predictions are not usable – this is reflected by missing values in the figure. We observe that this problem is mitigated by higher values for k, as we obtain usable results with higher C on all data sets if we choose $k = 2$ or $k = 3$, at the cost of increasing the QUBO size.

[15]`https://docs.ocean.dwavesys.com/projects/tabu/en/latest/`

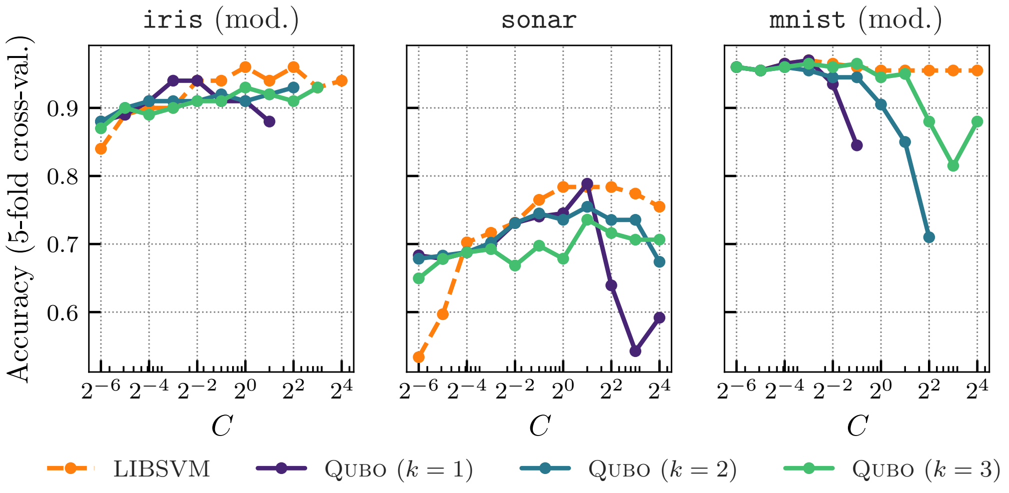


Figure 4.2.: 5-fold cross-validated prediction accuracies of the k-bit QUBO-SVM and LIBSVM on the three data sets listed in Table 4.1, using various values for C and k.

4.1.3. Concluding Remarks

It is possible to train SVM classifiers using a QUBO embedding that can be solved on quantum annealers to obtain the set of support vectors. The weightings $\boldsymbol{\alpha}$ from the dual SVM can be approximated to arbitrary precision by using k bits per weight, which leads to a QUBO formulation of size Nk, where N is the number of data points. We have seen that, while a higher k does not necessarily increase the classification accuracy, it allows for higher values of C by avoiding all-zero solutions to the QUBO problem. In general, however, the accuracy for lower C is competitive with LIBSVM, even for $k = 1$, which is surprising given that LIBSVM uses full floating-point precising.

The size of the QUBO problem is, at the current point of time, a limiting factor for the QUBO-SVM, as the number of variables is Nk. At the current point in time, this limits its use to data sets with only up to a few hundred data points. However, with further improvements in QA, it may be faster to train SVMs on large data sets using noise-free quantum annealers with a large number of qubits in the future, as classical SVM training algorithms such as SMO also scale polynomially in the number of data points [117]. Moreover, AQC on perfect hardware would be able to find the globally optimal set of support vectors, whereas many classical algorithms use local search methods. Therefore AQC has the potential to contribute to more accurate classification models.

4.2. Gate-Based SVM Deployment

Now that we have seen an example of how SVMs can be trained using QA, we focus on the deployment of already trained models. That is, given a feature vector $\varphi(\boldsymbol{x})$ and the weights $\boldsymbol{w}$, we want to compute the sign of the inner product, which allows us to obtain the model's prediction. This time, instead of QA, we use GQC to achieve this goal and construct the Real-part Quantum Support Vector Machine (RQSVM) circuit, which computes the required inner product through repeated measurement.

Since all operations in a quantum circuit must be unitary to ensure the preservation of quantum state properties, we will need a unitary representation of the model weights and the feature vector. To this end, we define a number of building blocks:

Definition 4.3 (Unitary Vector Embedding [2]). *Given a vector $v \in [-1,1]^d$, let $n = \lceil \log_2(d) \rceil$. The $2^n \times 2^n$-matrix $\Delta(v)$ whose elements are given by*

$$\Delta(v)_{j,k} = \begin{cases} \exp[-\mathrm{i} \arccos(v_j)] & \textit{if } j = k \textit{ and } j \leq d \\ -\mathrm{i} & \textit{if } j = k \textit{ and } j > d \\ 0 & \textit{otherwise (off-diagonal)} \end{cases}$$

is diagonal and unitary.

Unitarity follows from $|\exp[-\mathrm{i} \arccos(v)]| = 1$ for all $v \in [0,1]$. The definition of $\Delta(v)$ gives rise to an n-qubit quantum gate. In general, it is hard to factor arbitrary unitaries into elementary quantum gates, $\Delta(v)$ describes a *diagonal gate*, i.e., all off-diagonal entries of the unitary are 0. For such gates, a number of strategies have been discovered to implement them using low-dimensional basis gates [118, 119]. $\Delta(v)$ is our core building block that we use to represent $\boldsymbol{w}$ and $\varphi(\boldsymbol{x})$ in a quantum state.

As the next element, we define the following block diagonal matrix:

Definition 4.4 (Multiplexor [2]). *Given two unitary matrices $\boldsymbol{U}_1, \boldsymbol{U}_2$, their direct sum*

$$\boldsymbol{U}_1 \oplus \boldsymbol{U}_2 = |0\rangle\langle 0| \otimes \boldsymbol{U}_1 + |1\rangle\langle 1| \otimes \boldsymbol{U}_2$$

is unitary.

This notion is helpful to construct unitary operators that act simultaneously on independent sub-spaces of qubits. Decomposing $\boldsymbol{U}_1 \oplus \boldsymbol{U}_2$ into basis gates is straightforward as long as decompositions for $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$ are known [119]. Summation alone does generally not produce unitary matrices. However, due to the unitarity of $\boldsymbol{U}_1$ and $\boldsymbol{U}_2$, one can easily prove that $(\boldsymbol{U}_1 \oplus \boldsymbol{U}_2)(\boldsymbol{U}_1 \oplus \boldsymbol{U}_2)^\dagger = \boldsymbol{I}$ holds. As the dimension of the underlying quantum state is doubled, multiplexing requires an additional qubit. This *auxiliary* qubit will later be exploited to simulate a non-unitary operation.

Another component of the RQSVM circuit is the real-part extraction. Unitary vector embeddings of $\boldsymbol{w}$ and $\varphi(\boldsymbol{x})$ have a complex part that is required to make the corresponding operators unitary. However, when we compute the inner product of these operators, the result will not be equal to the desired inner product of weights and features, therefore we need to apply a real-part extraction to remove the complex part.

Definition 4.5 (Real-Part Extractor [2]). *] Given a diagonal unitary matrix $\boldsymbol{S}$, the unitary matrix*

$$R(\boldsymbol{S}) = (\mathbf{H} \otimes \boldsymbol{I}^{\otimes n})(\boldsymbol{S} \oplus \boldsymbol{S}^\dagger)(\mathbf{H} \otimes \boldsymbol{I}^{\otimes n})$$

allows us to apply $\operatorname{Re} \boldsymbol{S}$ *on some arbitrary quantum state $|\psi\rangle$, where $\mathbf{H}$ is the Hadamard gate and $\boldsymbol{I}$ is the identity matrix.*

The real part of any complex number $z \in \mathbb{C}$ can be written as $(z + z^*)/2$, as the complex parts are added to each other with opposite signs, canceling each other out. Similarly, for any $\boldsymbol{U} \in \mathbb{C}^{2^n \times 2^n}$, $(\boldsymbol{U} + \boldsymbol{U}^\dagger)/2$ extracts $\operatorname{Re} U$, such that it may act on the n-qubit state $|\psi\rangle$. On a gate-based quantum computer, we consider the multiplexed unitary $\boldsymbol{U} \oplus \boldsymbol{U}^\dagger$. Intuitively, by applying the Hadamard gate $\mathbf{H}$ to the auxiliary qubit $|a\rangle$, either $\boldsymbol{U}$ or $\boldsymbol{U}^\dagger$ will be applied to $|\psi\rangle$, each with probability 0.5. Depending on the measurement of $|a\rangle$, either the real or the imaginary part of $\boldsymbol{U}$ will be applied to $|\psi\rangle$. More precisely, we execute

$$R(\boldsymbol{U})(|0\rangle \otimes |\psi\rangle) = \frac{1}{2} \begin{pmatrix} \boldsymbol{U} + \boldsymbol{U}^\dagger & \boldsymbol{U} - \boldsymbol{U}^\dagger \\ \boldsymbol{U} - \boldsymbol{U}^\dagger & \boldsymbol{U} + \boldsymbol{U}^\dagger \end{pmatrix} (|0\rangle \otimes |\psi\rangle). \tag{4.4}$$

Clearly, when we eventually measure $|a\rangle = |0\rangle$, then the circuit successfully executed a non-unitary operation, namely $|\psi\rangle_+ = (\boldsymbol{U} + \boldsymbol{U}^\dagger) |\psi\rangle /2 = (\operatorname{Re} \boldsymbol{U}) |\psi\rangle$. On the other hand, when we measure $|a\rangle = |1\rangle$, then the output of the circuit is $|\psi\rangle_- = (\boldsymbol{U} - \boldsymbol{U}^\dagger) |\psi\rangle /2 = (\operatorname{Im} \boldsymbol{U}) |\psi\rangle$, and thus the imaginary part is extracted instead. Real-part extraction appeared before, e.g., in [120]. However, until now it has never been considered in the context of quantum SVMs or supervised QML in general.

One last building block is required to embed signed values into a quantum state:

Definition 4.6 (Sign Expansion [2]). *Given a vector $\boldsymbol{v} \in [-1, 1]^d$, the vector*

$$\boldsymbol{v}_\pm = (|0\rangle \otimes \boldsymbol{v}_-) + (|1\rangle \otimes \boldsymbol{v}_+)$$

is in $[0,1]^{2d}$, where $\boldsymbol{v}_+$ (respectively, $\boldsymbol{v}_-$) replaces all negative (respectively, positive) entries of $\boldsymbol{v}$ by 0.

A d-dimensional vector of signed numbers can be equivalently written as a $2d$-dimensional vector of unsigned numbers, with all positive and negative values stored separately, which is what Def. 4.6 implements.

Based on these building-blocks, we can construct the complete RQSVM circuit:

Theorem 4.1 (Real-Part Quantum SVM [2]). Let $\varphi : \mathbb{R}^m \to \mathbb{R}^d_+$, $\boldsymbol{w} \in \mathbb{R}^d$, $b > 0$ be the feature map and parameters of an SVM. Additionally, let

$$C_{\boldsymbol{w}}(\boldsymbol{x}) = R(W(\boldsymbol{w}/\|\boldsymbol{w}\|_\infty)\, U(\boldsymbol{x})), \tag{4.5}$$

where $W(\boldsymbol{w}) = \Delta(\sqrt{\boldsymbol{w}_\pm}) \oplus \Delta(\sqrt{\boldsymbol{w}_\pm})^\dagger$ and $U(\boldsymbol{x}) = R(\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm/\|\varphi(\boldsymbol{x})\|_\infty}))$. Here, square roots of vectors are computed element-wise. Model predictions are computed via

$$q_{\boldsymbol{w},b}(\boldsymbol{x}) = \operatorname{sign}(2^n \|\boldsymbol{w}\|_\infty \|\varphi(\boldsymbol{x})\|_\infty (\mathbb{P}_{\boldsymbol{w}}(j \leq 2d \mid \boldsymbol{x}) - 2\mathbb{P}_{\boldsymbol{w}}(j \leq d \mid \boldsymbol{x})) - b), \tag{4.6}$$

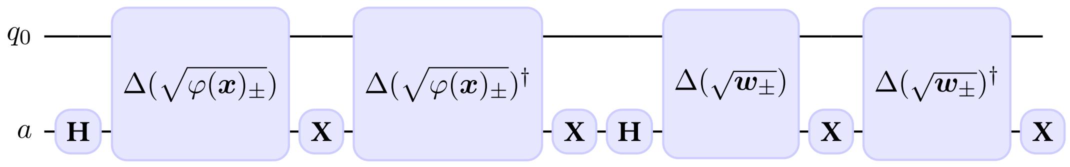

q_0
a
H
$\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm})$
X
$\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm})^\dagger$
X
H
$\Delta(\sqrt{\boldsymbol{w}_\pm})$
X
$\Delta(\sqrt{\boldsymbol{w}_\pm})^\dagger$
X

Figure 4.3.: Circuit diagram of $W(\boldsymbol{w})U(\boldsymbol{x})$ for a 2-dimensional RQSVM. Adapted from [2].

with $|\psi_{\text{in}}\rangle = |0\rangle^{\otimes 3} \otimes (H \otimes |0\rangle)^{\otimes n}$ and $n = \lceil \log_2(d) \rceil$. We find that $q_{\boldsymbol{w},b}(\boldsymbol{x}) = \hat{y}_{\boldsymbol{w},b}(\boldsymbol{x})$, i.e., the output of the RQSVM circuit $C_{\boldsymbol{w}}(\boldsymbol{x})$ is identical to the classical SVM with parameter vector $\boldsymbol{w}$ and feature vector $\varphi(\boldsymbol{x})$.

Proof. Let $c = (2^n \|\boldsymbol{w}\|_\infty \|\varphi(\boldsymbol{x})\|_\infty)^{-1}$. By Definitions 4.3, 4.4, 4.5, and 4.6 we have

$$\begin{aligned} \mathbb{P}_{\boldsymbol{w}}(j \mid \boldsymbol{x}) &= |\langle j| R(W(\boldsymbol{w}/\|\boldsymbol{w}\|_\infty)U(\boldsymbol{x})) |\psi_{\text{in}}\rangle|^2 \\ =& c \begin{cases} -\boldsymbol{w}_j \varphi(\boldsymbol{x})_j & \text{if } \boldsymbol{w}_j < 0 \text{ and } j < d \\ 0 & \text{if } \boldsymbol{w}_j \geq 0 \text{ and } j < d \\ \boldsymbol{w}_j \varphi(\boldsymbol{x})_j & \text{if } \boldsymbol{w}_j > 0 \text{ and } j > d \\ 0 & \text{if } \boldsymbol{w}_j \leq 0 \text{ and } j > d \end{cases} \end{aligned}$$

for $1 \leq j \leq 2d$. Thus,

$$\begin{aligned} & \mathbb{P}_{\boldsymbol{w}}(j \leq 2d \mid \boldsymbol{x}) - 2\mathbb{P}_{\boldsymbol{w}}(j \leq d \mid \boldsymbol{x}) \\ =& \sum_{j=1}^{2d} \mathbb{P}_{\boldsymbol{w}}(j \mid \boldsymbol{x}) - 2\sum_{j=1}^{d} \mathbb{P}_{\boldsymbol{w}}(j \mid \boldsymbol{x}) \\ =& \sum_{j=1}^{2d} |\langle j| C_{\boldsymbol{w}}(\boldsymbol{x}) |\psi_{\text{in}}\rangle|^2 - 2\sum_{j=1}^{d} |\langle j| C_{\boldsymbol{w}}(\boldsymbol{x}) |\psi_{\text{in}}\rangle|^2 \\ =& c\boldsymbol{w}^\intercal \varphi(\boldsymbol{x}) \; . \end{aligned}$$

Plugging this into Eq. (4.6) yields $q_{\boldsymbol{w},b}(\boldsymbol{x}) = \hat{y}_{\boldsymbol{w},b}(\boldsymbol{x})$ as desired. $\square$

The theorem states that the RQSVM circuit can mimic any classical SVM with a positive feature map, $\varphi : \mathbb{R}^m \to \mathbb{R}^d_+$. In practice, however, this is not a limitation, as any (bounded) feature map can be translated into the positive orthant [9].

An example of a complete RQSVM circuit for 2-dimensional inputs is shown in Fig. 4.3: Unitary embeddings (Def. 4.3) are represented by the Δ gates. Each Δ gate is responsible for storing the 2-dimensional SVM weights or feature vectors in an operator on the first qubit. The auxiliary qubit a controls whether each Δ gate is active. Each alternating sequence of Δ and $\mathbf{X}$ gates realizes a multiplexor (Def. 4.4). The first multiplexor applies either $\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm})$ or $\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm})^\dagger$ to the qubit q_0, based on the state of the auxiliary qubit a. The Hadamard gate brings a into a superposition state, allowing both operators, $\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm})$ and $\Delta(\sqrt{\varphi(\boldsymbol{x})_\pm})^\dagger$, to be applied to q_0 simultaneously. Depending on the state of a, either the real-part or the imaginary part is extracted (Def. 4.5). Since the Hadamard gate $\mathbf{H}$ is its own inverse, the second application of $\mathbf{H}$ returns the auxiliary qubit a to its initial state $|0\rangle$. Subsequently, $\Delta(\sqrt{\boldsymbol{w}_{\pm}})$ is multiplexed. The circuit requires $n = \lceil \log_2(d) \rceil + 3$ and hence $\mathcal{O}(\log_2(d))$ qubits. The three additional auxiliary qubits arise from two chained applications of the real-part extractor, as well as the sign expansion. When $\boldsymbol{w}$ is positive, sign expansion is not required, and the decision function simplifies to $q^{+}_{\boldsymbol{w},b}(\boldsymbol{x}) = \operatorname{sign}(2^n \mathbb{P}_{\boldsymbol{w}}(j \leq d \mid \boldsymbol{x}) - b)$.

We have no direct access to $|\psi_{\text{out}}\rangle$ on an actual quantum computer, as the dimension $|\psi_{\text{out}}\rangle$ is 2^n. Accordingly, reading out all entries of the quantum state vector would take an exponential amount of time and space. Instead, the probability $\mathbb{P}_{\boldsymbol{w}}(\cdot \mid \boldsymbol{x})$ is estimated from measurements of $|\psi_{\text{out}}\rangle$. The probability for an additive error of ε is in $\mathcal{O}(\exp(-N\varepsilon^2))$, which follows from Hoeffding's inequality [121]. The error approaches 0 in the limit of infinite quantum measurements.

Because the actual number of measurements is finite in practice, the output of the RQSVM will suffer from statistical noise. Various additional sources of noise introduced by NISQ devices affect the outcome of the computation further (compare Section 2.4.4). Even though it was shown theoretically that the RQSVM reproduces the classical SVM results, we are still interested in its performance in practice.

4.2.1. Evaluating the Performance of Quantum Inference

So far it was proven theoretically that the output of the proposed RQSVM converges to that of the classical SVM. While the model is sound in theory, we want to assess empirically how well the RQSVM captures the classical hyperplane in practice, and how it relates to a plain quantum SVM. As discussed in Section 2.4.4, practical QC faces many challenges, and we want to compare the performance of our proposed method as it is in theory and in practice. To this end, the following sections present a series of illustrative experimental results obtained from an exact QC simulation, and from a 27-qubit IBM Q Falcon r5.11 superconducting quantum processor. We investigate the effect of increasing the data dimensionality (i.e., the number of input features) on the model's performance in Section 4.2.2, and finally compare the RQSVM to an alternative Quantum SVM model in Section 4.2.3. For our results, we report the prediction accuracy as in Section 4.1.2.

In conventional Quantum SVMs, parameter training itself can only be carried out on a quantum computer, or using a quantum simulator, which becomes more and more inefficient with an increasing number of features. The RQSVM, on the other hand, can be trained entirely classically, without relying on a costly and memory-inefficient quantum simulation or access to a quantum processor. Thus, in an initial experiment, we learn the parameters of a linear SVM, again using LIBSVM (see Section 4.1.2). To this end, we create a data set with $N = 100$ data points, each sampled uniformly at random from $[0, 1]^2$. We extract the estimated parameters $\boldsymbol{w} = (3.914, 2.966)^\intercal$ and $b = 2.145$ from the learned model and load them into our quantum circuit. To better understand the impact of a noisy quantum computation, a comparison of the LIBSVM model, a stochastic quan-

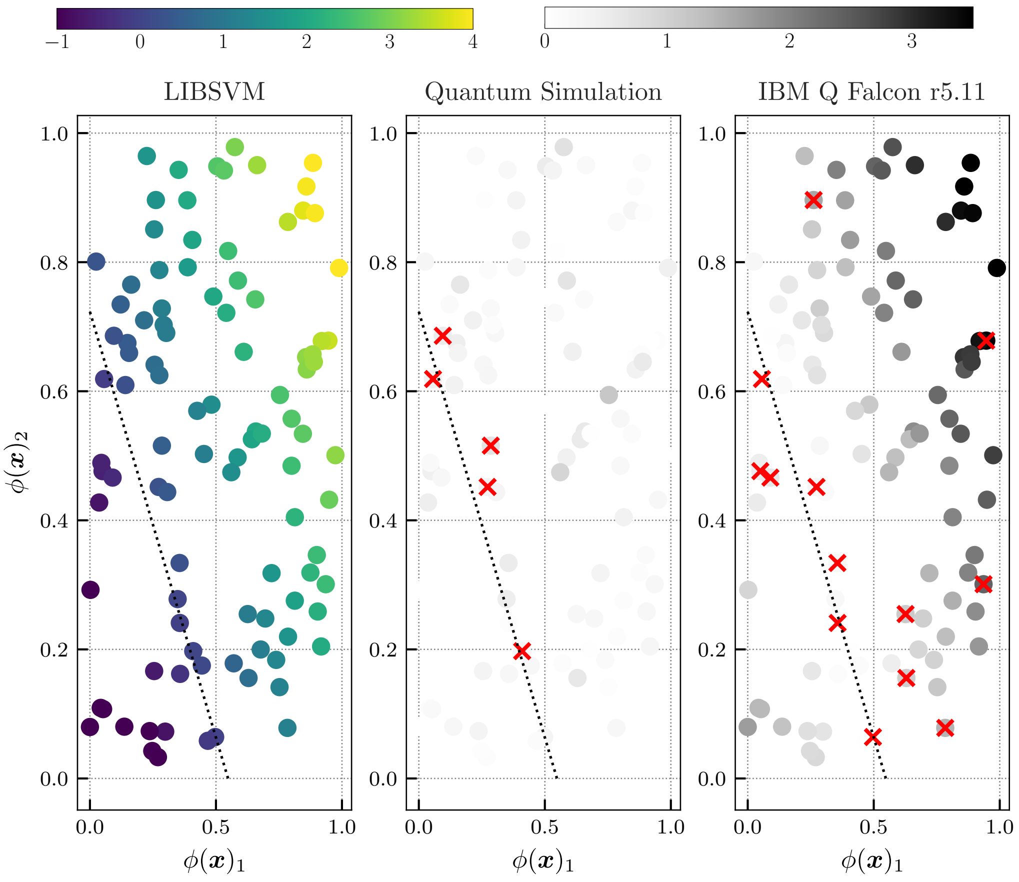


Figure 4.4.: An SVM with $\boldsymbol{w} = (3.914, 2.966)$ and $b = 2.145$ on a data set of $N = 100$ points, using three inference platforms LIBSVM, a quantum simulator, and a real quantum device. Red crosses indicate misclassified points. Source: [2].

tum simulation (including statistical noise caused by measurement), and results from an actual quantum processor is provided in Fig. 4.4: The dotted line is the separating hyperplane. The leftmost plot shows the results for a classical implementation of Eq. (2.11), where color indicates the value of $\boldsymbol{w}^\top \varphi(\boldsymbol{x}) - b$. For comparison, the center plot shows the predictions based on a simulation of $f(\boldsymbol{x}) = C_{\boldsymbol{t}}(\boldsymbol{x}) \left|\psi_{\text{in}}\right\rangle$ (see Theorem 4.1) for $M = 100$ measurements (or *shots*) per data point. Here, a darker color indicates the absolute error $|q_{\boldsymbol{w},b}(\boldsymbol{x}) - f_{\boldsymbol{w},b}(\boldsymbol{x})|$, and red crosses mark misclassified data points. The rightmost plot shows predictions computed from measurements taken on an IBM Falcon r5.11 superconducting quantum processor with matrix-free measurement mitigation [122]. Again, $M = 100$ shots are generated to estimate $\mathbb{P}_{\boldsymbol{w}}(j \leq d \mid \boldsymbol{x})$ for each of the 100 data points. Intuitively, we see from the simulation results (center) that statistical noise is responsible for deviations from the classical SVM (left) for data points that are close to the decision boundary. This type of noise is inherent to every sampling-based approach, and therefore to every method that relies on repeated quantum measurements. On the other hand, noise

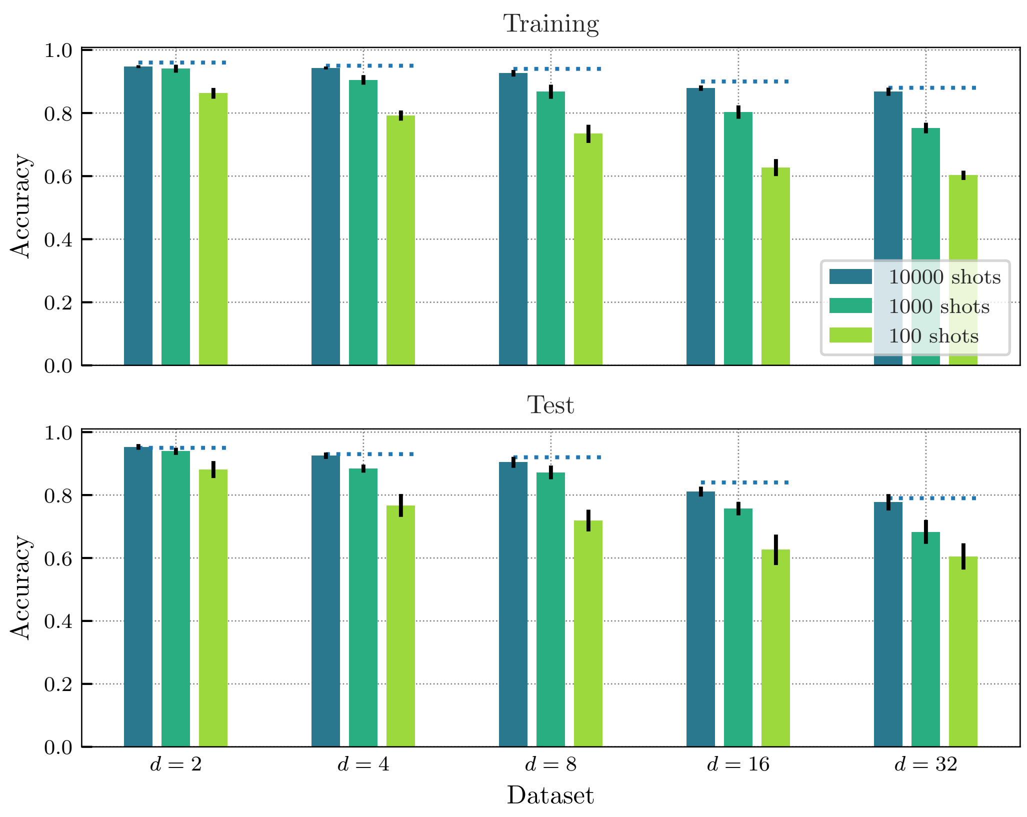


Figure 4.5.: Effect of an increased number of features on the prediction accuracy on synthetic train and test data. Error bars indicate the uncertainty caused by measurement noise. The dotted line shows the accuracy of the classical SVM. Source: [2].

that arises from the quantum hardware (right) results in deviations from the input model which are rather far away from the decision boundary. While it is currently possible to mitigate this type of error with [123, 122], eliminating it completely generally not possible. Upcoming generations of quantum computing hardware might allow a correction of errors [124], but these methods do not apply to current NISQ hardware. Nevertheless, the quantum hardware correctly reproduces 87 of 100 SVM predictions, which is an overwhelming result in the era of NISQ hardware.

4.2.2. Increasing the Number of Features

The previous results were obtained on a data set with only two features. Conventional Quantum SVM models scale well with the number of features, therefore the question of how the performance of the RQSVM model changes when we increase the number of input features arises naturally.

This time, we sample 300 data points consisting of d features for each $d \in \{2, 4, 8, 16, 32\}$. The number of data points is quite arbitrary; choosing a relatively low number of a few hundred points leads to quick SVM parameter training. Choosing d as powers of 2 allows us to observe the effect of dimensionality on increasing orders of magnitude. As input features, we use d-dimensional vectors sampled uniformly from $[0, 1]^d$. Next we draw a hyperplane through the center of the hypercube by sampling $\boldsymbol{w} \in [0, 1]^d$ uniformly from the set of vectors that sum to 1 and setting the bias $b = -\|\boldsymbol{w}\|_1/2$ so that the decision boundary runs through the middle of the unit hypercube. Labels are generated by setting $y_i = 1$ with probability $1/(1 + \exp(-50(\boldsymbol{w}^\intercal \boldsymbol{x} + b)))$, the intuition being that points very close to the hyperplane are randomly assigned to either $\{-1, +1\}$ based on this probability. This leads to a data set that is not perfectly linearly separable, requiring the use of slack variables. The resulting data sets for each d are split into 200 training and 100 test samples.

We proceed to train a classical SVM and perform a parameter search for the best regularization parameter C by trying all $C = 2^c$ for $c \in \{-8, \dots, 7\}$ and choosing the value that yields the highest accuracy. Next, we construct the RQSVM circuit and fix the weights to the vector $\boldsymbol{w}$ taken from the trained classical SVM, and run it repeatedly, taking 10^2, 10^3, and 10^4 measurements. For each model, we report the training and test accuracies.

Figure 4.5 shows the results of this experiment: The dotted line indicates the theoretical accuracy computed by the classical SVM, while the colored bars show the measured results for the different numbers of shots. As expected, the performance of the RQSVM approaches that of the classical SVM when the number of measurements is increased, and the amount of uncertainty descreases accordingly. Notably, already a relatively low number of 10^3 shots achieves near-optimal performance, which is particularly true for lower dimensions. More generally, we observe that the performance gap between the number of shots gets more pronounced with larger feature dimension. This is most likely due to the larger number of qubits requiring more samples to estimate $\mathbb{P}_{\boldsymbol{w}}(j \mid \boldsymbol{x})$ more accurately. However, this effect is less pronounced than we expected, indicating that the increase in dimension does not drastically affect the performance. Lastly, we observe a general drop in accuracy with higher dimensions, which also occurs in the classical SVM model and is most likely due to the fixed number of 200 training data points, which becomes increasingly insufficient to estimate sufficiently accurate SVM model parameters. This result once more shows that the RQSVM replicates the classical SVM's behavior faithfully up to statistical noise.

4.2.3. Comparison to a Conventional Quantum SVM

In our final experiment, we investigate the prediction quality of the RQSVM in comparison to a conventional Quantum SVM on a benchmark data set. The `digits` data set[16] contains small images of hand-written digits (0 to 9), similar to `mnist` used in Section 4.1.2.

[16]`https://archive.ics.uci.edu/dataset/80/optical+recognition+of+handwritten+digits` (last accessed June 3, 2025)

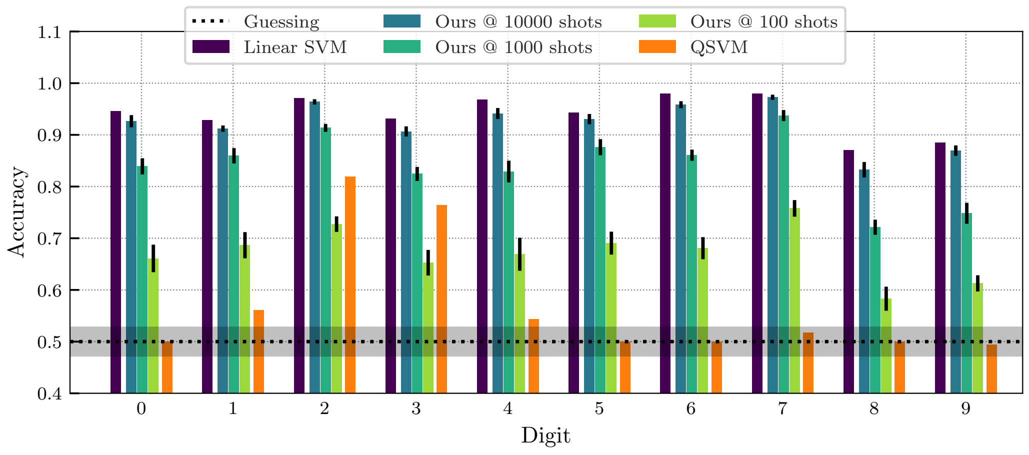


Figure 4.6.: Comparison of the per-class accuracy between random guessing (Bernoulli distributed), the classical SVM, a standard Quantum SVM with an noise-free quantum simulation, and the RQSVM with noisy simulation of 10^3, 10^4, and 10^5 shots on the `digits` data set. Error bars indicate the uncertainty that arises due to statistical (measurement) noise. Source: [2].

All images are gray-scale and have size 8×8 pixels, which yields 64 numerical features representing pixel brightness with values 0 to 16. In total, the set contains 1797 images, with about 180 images for each digit. For our experiment, we divide the pixel brightness values by 16 to normalize them to the range $[0, 1]$. As we did for `mnist`, to further reduce dimensionality we apply max-pooling to all 2×2 tiles of the images, reducing them to size 4×4 with 16 features. For each digit j, we select 174 images (the minimum number of instances available for each digit) of j and combine them with an equal number of randomly selected non-j images from the remaining data set. This leaves us with a balanced 1-vs-all classification data set for each digit, on which we train one RQSVM classifier each using the following strategy: We train a classical linear SVM and, as for the previous experiment, detect the best C, trying all $C = 2^c$ for $c \in \{-8, \dots, 7\}$, and save the parameters $\boldsymbol{w}$ and b that yielded highest accuracy. Then we build the RQSVM circuit with those fixed weights $\boldsymbol{w}$ and evaluate it for every image in the sub-data set with 10^2, 10^3, and 10^4 shots respectively.

As a baseline model to compare the RQSVM against, we train a conventional Quantum SVM without real-part extraction. For a fair comparison, the plain Quantum SVM has the same number of parameters as our RQSVM and uses the same feature map. The conventional Quantum SVM uses four entangling layers[17] with alternating rotation gates on each qubit, followed by **CX** gates between all adjacent qubits. The circuit parameters are trained using the ADAM optimizer [107] set to a learning rate of 0.1, which is run for 1000 iterations. To compute the gradients of the parametrized quantum circuit, the parameter

[17] `https://docs.pennylane.ai/en/stable/code/api/pennylane.BasicEntanglerLayers.html` (last accessed June 3, 2025)

shift rule is employed [125]. To eliminate the effect of noise for the conventional Quantum SVM, we use a noise-free statevector simulation, which is equivalent to obtaining an infinite number of shots.

The 1-vs-all classification accuracies for all classifiers are shown in Figure 4.6. As we have already shown in the previous experiment, we find that our RQSVM converges to the classical SVM's performance as the number of shots increases. With the data set's greater complexity compared to the synthetic data used in previous experiments in mind, 100 and 1000 shots result in significant performance penalties across almost all classes. Nevertheless, the RQSVM with 10^4 shots is always very close to the classical SVM. The conventional Quantum SVM is, however, not capable of reproducing its classical counterpart to a comparable degree, falling short of the RQSVM's performance using only 100 shots on most digits, except 2 and 3. Moreover, for at least 6 classes (digits 0 and 5-9), its accuracy is within a standard deviation of guessing class labels uniformly at random. This clearly shows that, given the same number of parameters, the RQSVM achieves much better classification performance compared to the conventional quantum model.

4.3. Concluding Remarks

As QC keeps evolving, it seems natural to ask under which conditions QML methods can be a useful extension to the set of well-established ML methods. While we know that there exists a very small set of classification problems which seem to be easier to learn with a quantum model [126, 127], the theoretical insights presented in this chapter prove that quantum circuits with $\log_2(d)$ qubits are capable of simulating classical SVMs with d-dimensional feature maps perfectly in the limit of infinite quantum measurements. Even for finite sample sizes of approximately 10^3 shots, our experimental results demonstrate that the RQSVM closely approaches the classical SVM and consistently outperforms conventional Quantum SVMs on the benchmark data we used. This number of quantum measurements is moderate, as practical implementations of quantum processing units typically allow for 10^4 or even 10^5 shots per circuit run.

The insights presented in this chapter allow us to apply theoretical findings by the ML community to the class of Quantum SVMs. While the broader problem of finding a quantum circuit that acts as a good feature map for a giving classification problem – or, for that matter, even deciding if a given circuit is a good feature map – remains unsolved, we have shown that the broad and well-understood class of (classical) SVMs is contained within the Quantum SVM framework, as quantum circuits can replicate them.

In the future, the RQSVM may prove beneficial in scenarios involving quantum data, where the input data comes not in form of a classical data set but as a set of quantum states, e.g., measured by quantum sensors, or produced as output by other quantum methods. Measuring such states to transform them into classical data and training a classical ML model would be inefficient, as we would need to store the empirical distribution classically, which requires exponential memory space. Instead, the quantum state could be processed directly by a quantum device. A direct implication of our findings is that such quantum states can be processed by a classical SVM that has been converted to an equivalent quantum circuit.

In summary, we have shown that both training and deploying SVMs is within reach of quantum computers, which opens up new avenues for both theoretical and applied research in QML, expanding our understanding of the landscape of Quantum SVMs.

Part III.

Classical Algorithms for Quantum Computing

5. Quantum Circuit Evolution

While the previous part has focused on applications of QC for the sake of supplementing ML methods, we now turn to the second aspect of this thesis: **How can methods and algorithms from classical computer science be used to supplement QC?**

With quantum devices still in the NISQ era (recall Section 2.4.4), we have to deal with a number of problems and limitations imposed by the imperfect hardware we have access to. Although quantum algorithms exist whose theoretical runtime guarantees supersede those of their classical counterparts [62], the noise inherent to NISQ machines largely prevents the application of well-known quantum algorithms with proven speedups, except for some toy examples. The Variational Quantum Eigensolver (VQE) introduced in Section 2.4.2 is more robust and better suited to the available hardware: Here, one iteratively optimizes a set of parameters with respect to their performance on a given cost function. Applications include, among others, ground state approximation [128, 65], simulation of imaginary-time evolution [129], and also QML [130]. However, NISQ limitations still present significant challenges to a practical application: They only allow for low circuit depths due to large error probabilities and short decoherence times, which multiply with every layer of gates (see Section 2.4.4).

In addition to practical limitations, the problem of barren plateaus [22] causes gradients of cost functions to become exceedingly small as the number of system qubits is increased, prohibiting gradient methods when dealing with a large number of parameters. In turn, this diminishes some of VQE's potential for problems of a practically relevant size [131]. To bypass such issues, we investigate EAs introduced in Section 2.2.2 for learning not only the parameters of circuits, but their overall structure, eliminating the need for gradient computation entirely.

In this chapter we evaluate how well evolutionary optimization is suited to the problem of learning quantum gate circuits, given an objective function that corresponds to the expectation value of a target Hamiltonian. More precisely, we define a mutation operator that uses *insertion*, *deletion*, *swapping*, and *modification* of circuit gates. The resulting specialized EA with this operator we call Evolutionary Circuit Learning (ECL).

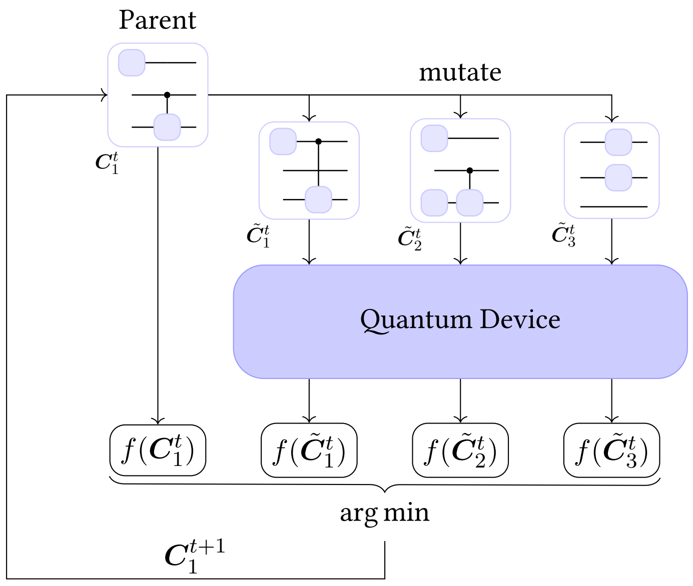


Figure 5.1.: Outline of evolutionary circuit discovery with $\mu = 1$ and $\lambda = 3$. Adapted from [6].

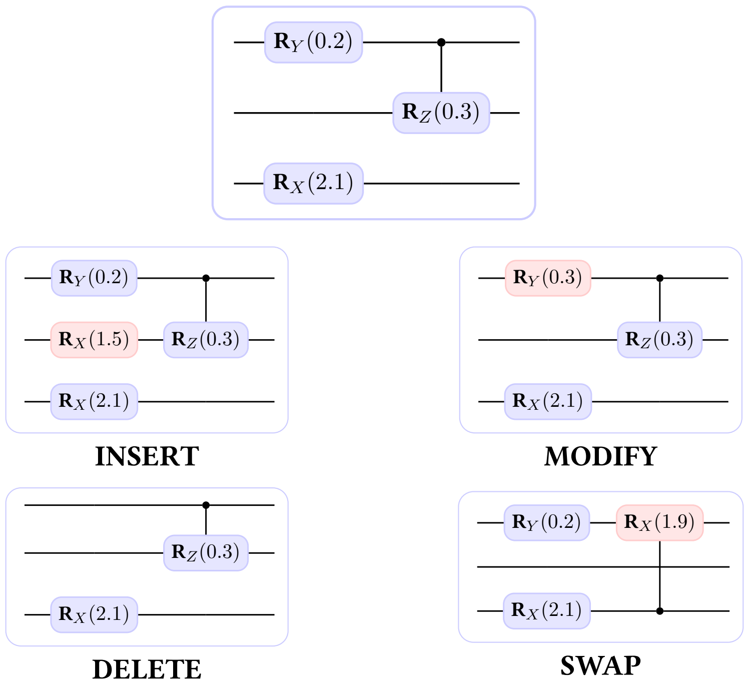


Figure 5.2.: Exemplary visualization of mutation actions performed by ECL. Adapted from [6].

This chapter is based on the publication [6]. The author of this thesis introduced the idea of using evolutionary optimization for circuit learning, defined the mutation operator, and implemented the optimization routine in Python. He formalized the the evolutionary algorithm and wrote the corresponding sections in the paper. He jointly designed and conducted the experiments, and developed the tools for analyzing the EA operations.

From Section 2.4 we recall that the Hamiltonian of a quantum system is an operator (that is, a complex-valued Hermitian matrix) corresponding to the total energy of that system. By testing the algorithm on Hamiltonians of varying difficulty, we show that particular operations become more important than others. In case of a local Hamiltonian, the algorithm tends to inserts gates in $\approx 7\%$ of all cases, while the other operations are not as useful. In contrast, for a Spin-Glass Hamiltonian, the success rate of the swapping operation is highest at about 11%, indicating that for more difficult problems gate insertion might be insufficient for successful optimization, and refining similar observations from literature [132].

Similarly to Chapter 4, we conduct experiments on real quantum hardware to see the impact of NISQ hardware on the performance of EA on quantum circuits. As we expect on NISQ hardware, convergence towards ground states prepared by non-local gates happens slower when IBM quantum backends are used. Finally, we compare the convergence speed to that achieved by GD methods and find that the evolutionary scheme outperform the gradient method in terms of the number of circuit evaluations, as sketched in Fig. 5.8. While this result depends on the quantum devices at hand and their properties (number of qubits, error rates, etc.), it shows that EAs are a viable option for quantum circuit construction.

Literature Overview The idea of VQE is the foundation for everything we discuss in this chapter, see Section 2.4.2. While variational approaches were developed concurrently [133], there are two broad categories of work connecting evolutionary algorithms and quantum computing: Quantum-inspired EAs for classical computers, and simulated quantum EAs.

Quantum-inspired EAs for classical computers simulate qubits, gates, superposition, and quantum measurement to solve various problems within the standard optimization framework, hoping to benefit from richer data representations enabled by quantum states and superposition [134]. The concept can be transfered to deep ANN architecture optimization, where it is able to yield effective yet simple CNNs [135]. The disadvantage of these techniques is the considerable computational cost of quantum simulations. For instance, the authors of [135] had twenty Nvidia K80 GPUs (using a total of 480 GB of GPU RAM) running for two days to perform their experiments.

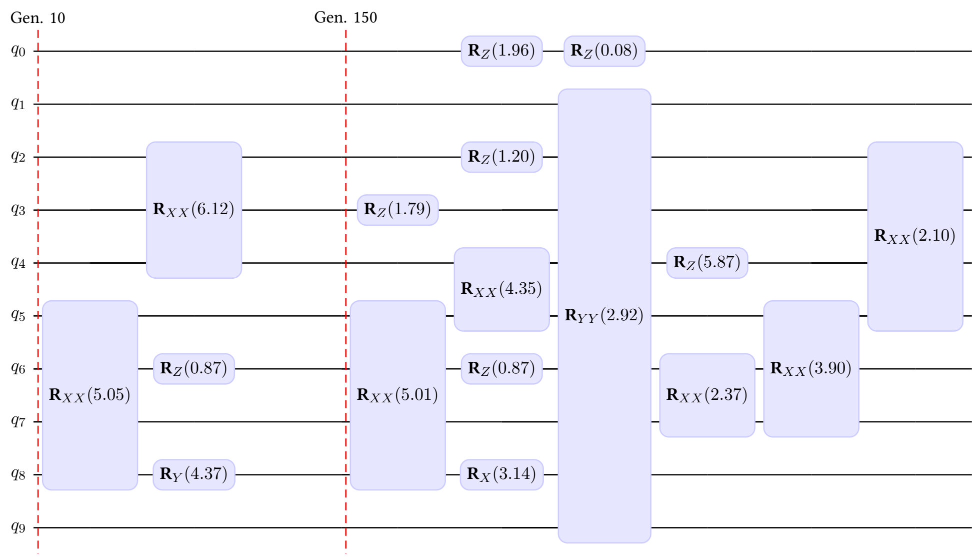


Figure 5.3.: Circuit evolution minimizing the transverse field Ising Hamiltonian. **Left:** After 10 generations. **Right:** After 150 generations. Gate parameters are rounded to two decimal places. Gates visually spanning multiple qubits are only applied to the uppermost and lowermost qubits. Adapted from [6].

Simulated quantum EAs utilize classical EAs in a simulated quantum computation environment. As an early success, such an algorithm was able to find a solution to Deutsch's problem [136]. More recently, a special Ising-type quantum computer was used to evolve multiple quantum gates in simulation [137].

Limited availability of working quantum hardware has long been a prohibitive factor for quantum evolutionary computing [138, 139]. However, the evolutionary approach has gained some traction, as it reduces the quantum computational overhead of exceedingly deep ansätze [63]. As discussed in Section 2.4.4, NISQ hardware limits the depth and complexity of circuits that can be executed successfully, which makes the prospect of keeping them small and simple promising. Particularly the Adapt-VQE algorithm [133] has shown promising results by alternating between optimizing the circuit ansatz and the parameter configuration. Another issue potentially solved by using non-gradient methods such as EAs is the phenomenon of vanishing gradients [140]. However, other works argue that the problem may persist [141]. Finally, additional problems like abrupt training transitions can arise [142].

5.1. Evolutionary Circuit Learning

To quickly recap Section 2.2.2, generic EAs iteratively work with a population of candidate solutions, and optimization is carried out over a number of generations; in each generation, $\mu > 0$ candidates that constitute the parent population produce an offspring population of $\lambda \geq \mu$ candidates by means of crossover and mutation; these operations are specific to the problem domain at hand. In our application, candidates are quantum circuits, and mutations cause small changes to the circuit, such as addition or removal of a gate, or nudging of a gate's parameter. We use plus selection, which exhibits elitism, ensuring that the overall best loss value always monotonically decreases over time. If none of the offspring individuals yields an improvement, the original parent population carries over to the next generation unchanged.

The classical Neuro-Evolution of Augmenting Topologies (NEAT) algorithm [143] adapts evolutionary algorithms for learning the structure of ANNs jointly with weight optimization: The initial parents have a minimal structure to avoid overly complex solutions. Due to the *competing conventions* problem [143] we choose not to include crossover: In ANNs, the order of neurons within hidden layers is arbitrary, as they can be permutated and still produce the same model output. Similarly, the construction of quantum circuits allows for many degrees of freedom. For example, the order of quantum gates acting on independent sub-systems is arbitrary (i.e., a convention) [62]. Consequently, combining parts of two individuals following different conventions tends do destroy the overall functionality.

The resulting EA we describe in the following evolves candidate solutions solely by applying a domain-specific mutation operator. To this end, we define a set of basis gates as the building blocks of the circuit architecture. As the circuits should be members of the entire unitary group over the chosen number of qubits, we give the algorithm access to a universal set of gates (cf. Section 2.4.5). Evolutionary optimization allows us to find a circuit $\boldsymbol{C} \in \mathcal{C}$ that approximates the best possible circuit with respect to some cost function $f : \mathcal{C} \to \mathbb{R}$, where $\mathcal{C}$ represents the set of all possible circuits. In the following sections, we refer to our method as Evolutionary Circuit Learning (ECL).

5.1.1. A Universal Gate Set

Assume we are constructing circuits with $n \geq 2$ qubits. The circuits we consider are constructed from a set $\mathcal{G}$ of gates of the form

$$\mathbf{G}(\boldsymbol{U}, \theta) = \exp\left[-\mathrm{i}\frac{\theta}{2}\boldsymbol{U}\right], \tag{5.1}$$

where $\mathbf{U}$ is a base unitary from a pre-defined set $\mathcal{U}$, and $\theta \in [0, 2\pi)$ is a rotation angle. The exponential function represents the matrix exponential here. Our base unitaries $\mathcal{U}$ are the union of single-qubit operations $\mathcal{U}_{\text{single}}$ and two-qubit operations $\mathcal{U}_{\text{two}}$, where

$$\mathcal{U}_{\text{single}} = \bigcup_{1 \leq i \leq n}^{n} \{\boldsymbol{\sigma}_i^{(x)}, \boldsymbol{\sigma}_i^{(y)}, \boldsymbol{\sigma}_i^{(z)}\}, \tag{5.2}$$

$$\mathcal{U}_{\text{two}} = \bigcup_{1 \leq i < j \leq n} \{\boldsymbol{\sigma}_i^{(x)}\boldsymbol{\sigma}_j^{(x)}, \boldsymbol{\sigma}_i^{(y)}\boldsymbol{\sigma}_j^{(y)}, \boldsymbol{\sigma}_i^{(z)}\boldsymbol{\sigma}_j^{(z)}\}, \tag{5.3}$$

$$\mathcal{U} = \mathcal{U}_{\text{single}} \cup \mathcal{U}_{\text{two}}. \tag{5.4}$$

This effectively generates all possible single-qubit ($\mathbf{R}_X$, $\mathbf{R}_Y$, $\mathbf{R}_Z$) and two-qubit ($\mathbf{R}_{XX}$, $\mathbf{R}_{YY}$, $\mathbf{R}_{ZZ}$) rotation gates, applied to each combination of qubit indices.

Proposition 5.1 (Universality of $\mathcal{G}$ [6]). The gate set $\mathcal{G}$ as defined above is universal.

Proof. From [144] we know that $\{\mathbf{R}_Y, \mathbf{R}_Z, \mathbf{CX}\}$ is a universal gate set. One has

$$\mathbf{CX} = (\mathbf{R}_Y(-\frac{\pi}{2}) \otimes \boldsymbol{I}_2)(\mathbf{R}_{XX}(-\frac{\pi}{2}))(\mathbf{R}_X(\frac{\pi}{2}) \otimes \mathbf{R}_X(-\frac{\pi}{2}))(\mathbf{R}_Y(\frac{\pi}{2}) \otimes \boldsymbol{I})(\mathbf{S}(\frac{7\pi}{4}) \otimes \mathbf{S}(\frac{7\pi}{4}))$$

with the global phase shift gate $\mathbf{S}(\delta) = \exp[\mathrm{i}\delta \boldsymbol{I}]$ and the identity gate $\boldsymbol{I} = \mathbf{R}_Z(0)$, where $\mathbf{R}_X(\delta) = \mathbf{R}_Y(\pi/2)\,\mathbf{R}_Z(\delta)\,\mathbf{R}_Y(-\pi/2)$ and $\delta \in [0, 2\pi)$. Since the global phase shift does not affect measurement outcomes, $\mathcal{G}$ is indeed universal. □

The native gate set of IBM quantum devices is in fact $\mathcal{G}_{\text{IBMQ}} = \{\boldsymbol{I}, \mathbf{R}_Z, \mathbf{X}, \sqrt{\mathbf{X}}, \mathbf{CX}\}$, where $\mathbf{X} = \mathbf{R}_X(\pi)\,\mathbf{S}(\pi/2)$ and $\sqrt{\mathbf{X}} = \mathbf{R}_X(\pi/2)\,\mathbf{S}(-7\pi/4)$. In order to run a circuit consisting of gates from the set $\mathcal{G}$ on IBM hardware, the circuit has to be transpiled to only use gates from $\mathcal{G}_{\text{IBMQ}}$ (see Section 2.4.5).

5.1.2. Mutating Quantum Circuits

The cost value of a circuit is defined by the expectation with respect to some target Hamiltonian $\boldsymbol{H}$ with $|\psi_0\rangle$ as initial state:

$$f(\boldsymbol{C}) \equiv \langle\psi_0|\, \boldsymbol{C}^\dagger \boldsymbol{H} \boldsymbol{C}\, |\psi_0\rangle\,. \tag{5.5}$$

Finding a circuit $\boldsymbol{C}_{\text{opt}}$ that minimizes f is an optimization task over the search space $\mathcal{U}$, defined through a mix of discrete and continuous values, namely unitaries $\mathbf{U} \in \mathcal{U}$ with their respective qubit indices, and real-valued parameters θ.

While gradient methods are restricted to optimize the real-valued parameters θ on fixed circuit layouts, they cannot efficiently learn the overall circuit structure, since adding or removing gates are both non-differentiable operations. Indeed, the gates in $\mathcal{G}$ are certainly differentiable w.r.t. θ around $\theta = 0$, and applying a rotation gate with angle parameter 0 has the same effect as applying no gate at all. Therefore, insertion and deletion of gates can, in principle, be simulated by gradient-based methods. However, this would require a very large initial circuit that contains all attainable circuits as a sub-structure – a strategy that should be avoided due to the limited depth allowed on NISQ devices.

EAs with elitist selection as described in Section 2.2.2 can deal with non-differentiable and even non-continuous search spaces, because they only require some mutation operator $\mathfrak{m}$ that takes a circuit $\boldsymbol{C}$ as input, applies random changes to it, and produces a slightly altered circuit $\tilde{\boldsymbol{C}}$. More formally, we can interpret a circuit as a sequence of gates, $\boldsymbol{C} = (\mathbf{G}_1, \ldots, \mathbf{G}_L)$, for some $L \in \mathbb{N}$ and $\mathbf{G}_i \in \mathcal{G}\ \forall\, i \in \{1, \ldots, L\}$. The set of all circuits can be represented as $\mathcal{G}^+ = \bigcup_{L\in\mathbb{N}} \mathcal{G}^L$. Then, $\mathfrak{m}(\boldsymbol{C})$ is a random variable over $\mathcal{G}^+$. If $\boldsymbol{C}$ has support everywhere on $\mathcal{G}^+$, i.e. $\mathbb{P}(\mathfrak{m}(\boldsymbol{C}) = \tilde{\boldsymbol{C}}) > 0$ for all $\tilde{\boldsymbol{C}} \in \mathcal{G}^+$, then the EA is guaranteed to converge to the global optimum [145].

Assume we have μ parent circuits $\boldsymbol{C}_1^t, \ldots, \boldsymbol{C}_\mu^t$ in generation t. To find the parents of the next generation $t+1$, we perform the following steps:

1. Sample λ offspring circuits $\tilde{\boldsymbol{C}}_1^t, \ldots, \tilde{\boldsymbol{C}}_\lambda^t$ by mutating random parents, i.e., $\tilde{\boldsymbol{C}}_i^t \sim \mathfrak{m}(\boldsymbol{C}_j^t)$ for every $i \in \{1, \ldots, \lambda\}$, and for some randomly chosen $j \in \{1, \ldots, \mu\}$ per i. Ideally, $\lambda \geq \mu$ and every parent circuit is mutated at least once to preserve diversity in the population.
2. Sort both the parent and offspring circuits by the loss function f.
3. Take the μ circuits of lowest loss value and assign to them $\boldsymbol{C}_1^{t+1}, \ldots, \boldsymbol{C}_\mu^{t+1}$.

This process is repeated until some convergence criterium is met, e.g.,

- no more changes occur, i.e., for a given number of generations $\tau > 0$ we find
$$\{\boldsymbol{C}_i^{(t+\tau)}\}_{1 \leq i \leq \mu} = \{\boldsymbol{C}_i^{(t+\tau)}\}_{1 \leq i \leq \mu},$$
- the loss value does not decrease over τ generations, i.e.,
$$\min_{1 \leq i \leq \mu} f(\boldsymbol{C}_i^{t+\tau}) = \min_{1 \leq i \leq \mu} f(\boldsymbol{C}_i^t),$$
- some budget is depleted, e.g., a fixed maximum number of computations on a quantum device.

See Fig. 5.1 for a schematic outline of the Evolutionary Circuit Learning (ECL) algorithm. Note again that this setup is different to the framework of regular VQE [67], which assumes a fixed circuit $\boldsymbol{C}(\boldsymbol{\theta})$ with exposed parameters $\boldsymbol{\theta}$, and uses a gradient method to optimize f w.r.t. $\boldsymbol{\theta}$, leaving the circuit structure itself unchanged.

For our experiments, we defined the mutation operator $\mathfrak{m}$ as a two-level random process, consisting of *(i)* randomly choosing an action from a list of options, *(ii)* sampling the parameters for the chosen action.. The parametrized action is then applied to the circuit at hand. The possible actions (with their respective occurrence probabilities in parentheses) are:

- **INSERT** (50%): Sample unitary $\mathbf{U} \in \mathcal{U}$ and parameter $\theta \in [0, 2\pi)$ uniformly and insert the corresponding gate at a random position.
- **DELETE** (10%): Delete gate at a random position from the circuit.
- **SWAP** (10%): Combination of DELETE and INSERT at the same randomly chosen position.
- **MODIFY** (30%): Modify parameter of randomly chosen gate according to $\theta \mapsto \theta + \epsilon$ with $\epsilon \sim \mathcal{N}(0, 0.1)$.

The probabilities were found to perform best in a range of preliminary experiments. See Fig. 5.2 for a visualization of the four actions, and Fig. 5.3 for a larger example taken from one of our experiments.

With a probability of 10%, we repeat this entire mutation process after each action, leading to an expected number of $10/9 = 1.\overline{1}$ actions per mutation, the probability for 2 actions being about 9%, for 3 actions about 0.9% and for k actions $0.1^{k-1} \cdot 0.9$ in general. This scheme enables the mutation to perform arbitrarily large jumps in search space with positive probability, avoiding getting stuck in a local optimum indefinitely.

5.2. Experimental Evaluation

In this section, we evaluate the performance of ECL on standard VQE benchmark problems, both in simulations and on actual quantum devices. To this end, we use Qiskit to define and modify quantum circuits in Python, simulate them classically, and send them to IBM quantum devices (see Section 2.4.5).

For our experiments, ECL is implemented to perform an $(1+4)$-EA with the special multi-level mutation strategy described above. We found the choice of $\lambda = 4$ to be a suitable compromise between population diversity and speed. The optimization run starts with a minimal random circuit, consisting of a single gate with a uniformly sampled parameter. From this initial parent circuit, we make four copies and mutate them independently. The resulting offspring circuits are sent to the IBM backend. From the measurement results we derive a loss value, which we explain in detail in Section 5.2.1. The entire population is then sorted by loss value, and the circuit with lowest loss becomes the new parent for the next generation. This process is repeated, with the parent's cost value monotonically decreasing, approaching the global optimum.

We append rotation layers to the circuit to measure expectations w.r.t. the Pauli terms required by the Hamiltonian. For consistency, we leave most of the algorithm's hyperparameters (population sizes μ and λ, selection strategy, mutation action probabilities, etc.) unchanged for the majority of our experiments and choose $n = 10$ qubits. In every generation, we thoroughly collect data about the evolution process. This allows us to extract valuable information, such as loss value development over the generations, and success rates of mutations actions, which we can analyze later.

5.2.1. Target Hamiltonians

As shown in Eq. (5.5), the cost function that we use for ECL is the expected energy w.r.t. a target Hamiltonian. We expect that Hamiltonians whose ground states are highly entangled are more difficult to to construct. Intuitively, this can be explained by the fact that the qubits in non-entangled states (i.e., product states) can be optimized independently, while entanglement requires more intricate operations acting on larger sub-systems, which involve two-qubit gates. As an additional detrimental factor, two-qubit gates are more error-prone on NISQ hardware[18]. This should be reflected in differences in performance

[18]IBM reports gate errors of all their devices on their quantum computing platform `https://quantum.ibm.com/services/resources` (last accessed June 3, 2025); two-qubit gate errors are typically around an order of magnitude higher than single-qubit gate errors.

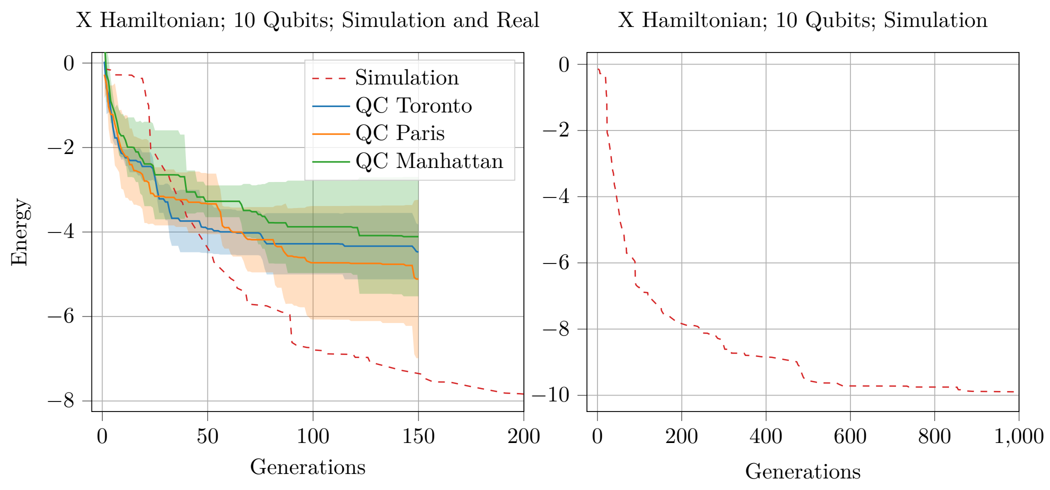


Figure 5.4.: Performance of ECL on $\boldsymbol{H}_{\text{local}}$ with $n = 10$, using real quantum backends Toronto, Paris, and Manhattan, and a simulator. The mean over 5 runs is shown. Lower is better. Source: [6].

between experiments on a simulator versus on actual quantum computers. To increase difficulty gradually, we transition in three steps from a local problem to an Ising spin-glass model only consisting of non-local terms, allowing us to investigate *(i)* our algorithm's capability to optimize increasingly difficult problems, and *(ii)* its performance on real quantum devices.

Local Hamiltonian As a sanity check, we first consider a local problem

$$\boldsymbol{H}_{\text{local}} = \sum_{i=1}^{n} \boldsymbol{\sigma}_i^{(x)}, \tag{5.6}$$

with $\boldsymbol{\sigma}_i^{(x)}$ being the Pauli-X operator acting on qubit i. Clearly, the ground state to this Hamiltonian is given by the n-qubit product state $|+\rangle^{\otimes n}$, which is a product state without any entanglement. For this relatively simple problem, we expect stable convergence in simulation and real hardware. We start with $|\leftarrow\rangle^{\otimes n}$ as the initial state, which is a ground state of $(\boldsymbol{\sigma}^{(y)})^{\otimes n}$.

Transverse-Field Ising Model (TFI) Next, we consider a 1D spin-chain with correlation in the Z-component, and a transverse magnetic field with X-axis orientation [146], modeled by the Hamiltonian

$$\boldsymbol{H}_{\text{TFI}} = -J \sum_{i=1}^{n-1} \boldsymbol{\sigma}_i^{(z)} \boldsymbol{\sigma}_{i+1}^{(z)} - J \boldsymbol{\sigma}_n^{(z)} \boldsymbol{\sigma}_1^{(z)} - h \sum_{i=1}^{n} \boldsymbol{\sigma}_i^{(x)}. \tag{5.7}$$

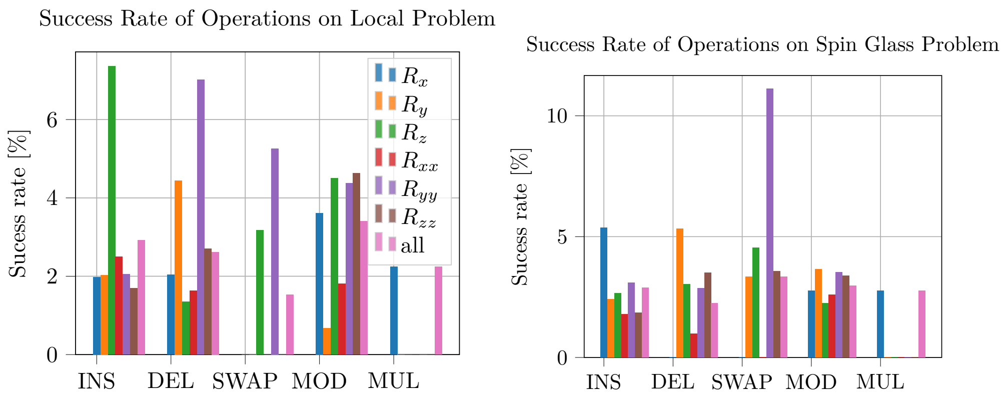


Figure 5.5.: Success rate of various mutation actions for each type of gate, averaged over all runs. **Left:** Local Hamiltonian $\boldsymbol{H}_{\text{local}}$ (Eq. (5.6)). **Right:** SK Hamiltonian (Eq. (5.8)). MUL indicates that a series of multiple operations was successful. Source: [6].

For our purposes, we choose $J = 1$ and $h = 1$, opting for anti-ferromagnetic behavior of this Hamiltonian. Note that Eq. (5.7) contains $\boldsymbol{H}_{\text{local}}$ as a sub-term, superimposing the component requiring entanglement on top. The TFI model is known to exhibit local minima, causing purely gradient-based methods to fail [147]. Here, ECL may be at an advantage. For this problem we again start with state $|{\leftarrow}\rangle^{\otimes n}$.

Sherrington-Kirkpatrick model (SK) The SK model simulates the behavior of a frustrated spin-glass [148] and was previously used as a benchmark model in quantum computing experiments [149, 128]. The model is given by the Hamiltonian

$$\boldsymbol{H}_{\text{SK}} = \sum_{1 \leq i < j \leq n} J_{ij} \boldsymbol{\sigma}_i^{(z)} \boldsymbol{\sigma}_j^{(z)}, \tag{5.8}$$

where J_{ij} are uniformly drawn from $\mathbb{S}$ for each pair i, j. For every run, the J_{ij} values are re-sampled. Note that $\boldsymbol{H}_{\text{SK}}$ consists exclusively of correlation terms, which is why we expect this optimization problem to constitute a considerably harder problem than the previous experiments. For this problem, we start with local $\boldsymbol{\sigma}^{(x)}$-eigenstates on all qubits, namely $|+\rangle^{\otimes n}$.

For our experiments, we used the IBM backends Manhattan, Toronto and Paris, alongside the Qiskit state vector simulator. For all circuit we use $n = 10$ qubits. Figures 5.6 and 5.7 show the lowest energy value in each generation and for each platform; the right-hand side plot shows only the simulation runs for more generations, to get a more complete picture of the convergence behavior. The loss function value (i.e., the expected energy of the resulting quantum states after applying the circuit w.r.t. the target Hamiltonian) is plotted as mean and variance of 5 evolution runs. Additionally, to gain more insight into the algorithm's choices, we recorded which gates and evolutionary operations contributed positively to the optimization process, shown as a histogram in Fig. 5.5. Overall, we find our algorithm to perform well on all posed problems, converging to a circuit

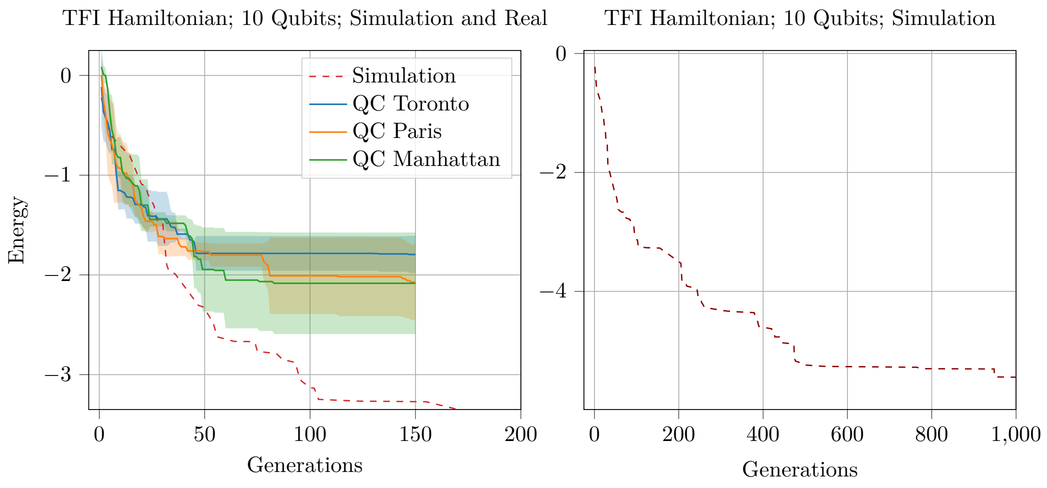


Figure 5.6.: Performance of ECL on quantum backends Toronto, Paris, and Manhattan, and a simulation for the TFI Hamiltonian on 10 qubits. All runs are shown as a mean of multiple runs. Lower is better. Source: [6].

with relatively low energy value. The histograms indicate intelligent circuit design both in gate and operation choice, mostly reflecting our intuitions about which rotations are useful for certain tasks. We find all operations to provide a non-negligible number of useful additions to the optimization, and we observe distinct differences in the operations used between the Hamiltonians. While for the local problem, insertion of $\mathbf{R}_Z$ gates and removal of entangling $\mathbf{R}_{YY}$ gates seems to be a successful strategy, for the SK problem, swapping gates with $\mathbf{R}_{YY}$ has a higher success rate. Generally, we observe a slight performance reduction on quantum hardware, particularly when multi-qubit gates are required, which suffer most from NISQ-related problems (cf. Section 2.4.4). The convergence rate of the simulation is faster, suggesting that the combination of sampling noise and NISQ-related noise make it harder for the EA to find improvements.

The results for $\boldsymbol{H}_{\text{local}}$ are shown in Fig. 5.4. In this experiment we optimize a sum of local $\boldsymbol{\sigma}^{(x)}$ terms over all qubits. The ground state of this Hamiltonian is the n-qubit product state $|-\rangle^{\otimes n}$ which can be prepared by local operations. Since the problem requires no entanglement between the qubits we expect relatively fast convergence for both the simulation and the real runs. Indeed we and observe so in Fig. 5.4 with all runs steadily converging to the global minimum. The convergence is steeper in simulation, which is probably due to sampling noise when evaluating the Hamiltonian approximately from a fixed number of measurements, whereas the simulator operates on the true complex-valued quantum states and thus computes the exact expectation value. The initial phases of the algorithm show a larger similarity between simulation and real hardware compared to the optimization after generation 50. Since for this experiment we chose the $\boldsymbol{\sigma}^{(y)}$-eigenstate on all qubits as the initial state, we expect rotations around the z-axis (i.e., $\mathbf{R}_Z$ gates) to be particularly useful. This intuition is confirmed by the analysis of success rates in Fig. 5.5, where we find that $\mathbf{R}_Z$ is indeed the rotation operation that sur-

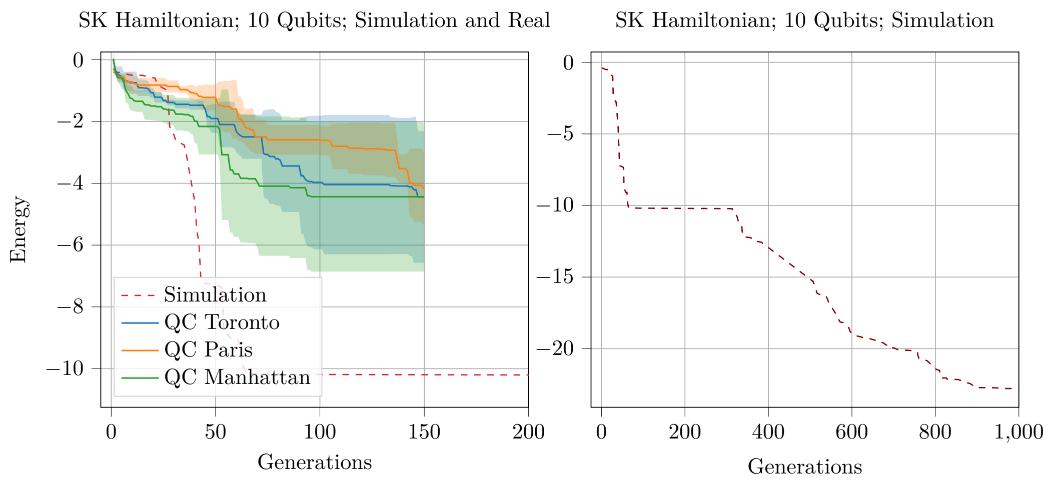


Figure 5.7.: Performance of ECL on quantum backends Toronto, Paris, and Manhattan, and a simulation for the SK Hamiltonian on 10 qubits. All runs are shown as a mean of multiple runs. Lower is better. Source: [6].

vived selection most frequently by a large margin. The figure shows further that modifications to z-rotations and replacements in favor of z-rotations are preferred over gate deletions.

Regarding the TFI model, shown in Fig. 5.6, we find that the simulated runs show reliable convergence, even for a relatively small number of offspring created per generation. Similarly to the local Hamiltonian, the runs on real hardware show a slower convergence than the simulated run. Gate noise of deep candidate circuits might be a limiting factor here.

Finally, the results of the SK Hamiltonian optimization are shown in Fig. 5.7. Again, the early generations make quick progress and show even faster convergence than the simulation. However, we observe generally slower improvement, and longer stretches without any progress. These stretches are also present in the simulation case, indicating that the required mutations to the candidate circuits are more specific, and thus unlikely. Unfortunately, the required number of generations to reach a good solution seems to be much higher than what we were able to perform, exceeding our feasible quantum computation time. Simulation, however, progresses after a substantial plateau until around 300 generations and reaches an energy value of about -23 at 1000 generations. However, Fig. 5.5 (right) gives some insight into the algorithm's gate and operation preferences during the optimization of $\boldsymbol{H}_{\text{SK}}$, which is notably different to the previously discussed local problem. The mutation operations that tend to be most successful seem to be more cautious in adding gates to the circuit. A relatively low frequency of gate insertions in comparison to gate swapping indicates that the circuit requires very specific gates in the right places. For this problem we start with local $\boldsymbol{\sigma}^{(x)}$-eigenstates in all qubits, making $\mathbf{R}_{YY}$ gates

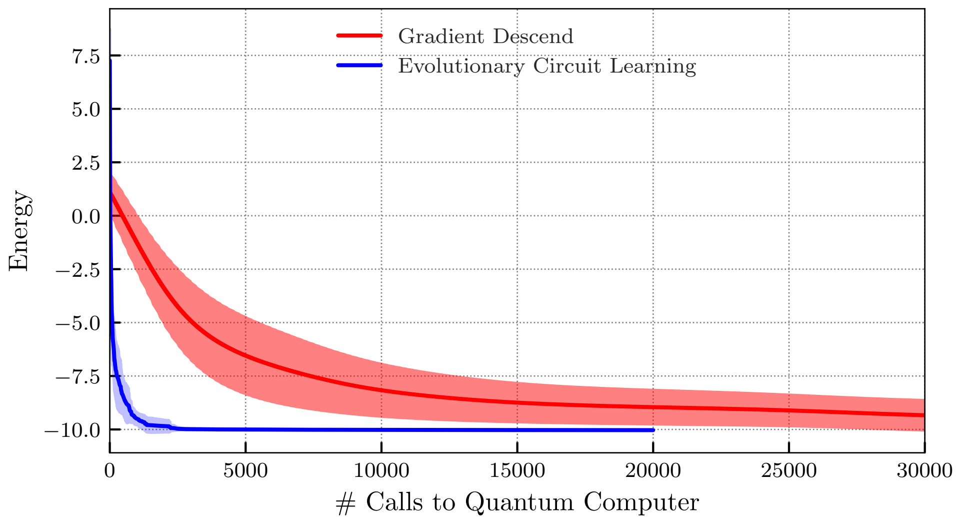


Figure 5.8.: Expected energy versus calls to a quantum computer for GD and ECL, minimizing an 8-qubit transverse field Ising Hamiltonian. Source: [6].

particularly useful, which we see confirmed by such gates being added predominantly to the circuit via swapping.

5.2.2. Comparing Quantum Circuit Evaluations

As a last comparison, we want to compare the number of calls to the quantum computer that ECL requires compared with GD. To this end, we minimize the expected energy of an 8-qubit transverse-field Ising Hamiltonian. GD starts with a random Pauli 2-design ansatz with 40 parameters, computing derivatives via parameter shifts [125]. These derivatives are used for parameter updates with a fixed learning rate $\eta = 0.01$ (cf. Section 2.2.1). For ECL, the usual $(1 + 4)$ EA using the gate set $\mathcal{U}$ is used, as before. Every circuits is initialized with a random gate. Five runs are performed in total, yielding mean and standard deviation over the random ansatz and initial parameters for GD, and over the initial gates and the particular mutations performed by ECL. To eliminate measurement noise, a noise-free state vector simulator was used.

Results are shown in Fig. 5.8: ECL shows a much faster convergence to a low-energy solution than GD in terms of the number of calls to the quantum device. The reason for this behavior is the necessity of GD to perform two calls per parameter, as the gradient w.r.t. every single parameter has to be evaluated separately. In contrast, ECL performs only one call for every offspring in each generation, i.e., 4 calls per generation in this experiment.

This result shows that ECL makes more efficient use of quantum resources than GD, which scales unfavorably with the number of parameters: While for each additional parameter, GD requires two more evaluations of the expected energy w.r.t. to the Hamiltonian, the number of calls to the quantum backend stays constant for ECL, saving on quantum resources and energy.

5.3. Concluding Remarks

In this chapter we have seen an algorithm based on an EA that circumvents some of the disadvantages of VQC: In an iterative process, this method creates a set of quantum circuits from a parent circuit by mutation, and subsequently selects the best performing candidate as parent for the next generation. We tested this algorithm called ECL on three Hamiltonians of varying difficulty, both in simulation and on actual quantum hardware. On all of our benchmark problems, ECL is reliably able to improve circuits according to the objective. The performance is decreased in experiments on real quantum devices, as is expected with NISQ devices.

As we assumed from the beginning, we found that the algorithm exhibits the following properties: A non-local Hamiltonian requiring entanglement impedes optimization, and as such, convergence on local problems proceeds faster. Between our experiments using $\boldsymbol{H}_{\text{local}}$ (with only single-qubit terms) and $\boldsymbol{H}_{\text{SK}}$ (with only ZZ-correlation terms), this increase in hardness is reflected in our results. A high number of correlation terms in the Hamiltonian implies that more multi-qubit gates are required for ground state preparation. Creating circuits with these multi-qubit gates in the right place by mutation has a lower probability compared to single-qubit gates. In addition, such gates are more affected by noise, which we observed in our experiments as a slowing down of runs on quantum hardware compared to simulation. Our statistics indicate that the circuit design is indeed somewhat intelligent in that not only useful gates are noticeably preferred, but also are redundant gates removed over time. The ability of ECL to remove or swap redundant gates, making the resulting circuit more compact and adapted to the Hamiltonian at hand, is an advantage over gradient methods commonly used for VQE.

While performing the experiments, we recorded the success rates of mutation actions and their associated gate types. The results demonstrate that all mutation operations (insertion, deletion, swapping, and modification) have considerate success rates across our experiments, i.e., no operation is completely redundant, and they all work together to generate useful alterations of the parent circuit, depending on the target problem. This suggests that the application of evolutionary strategies to quantum circuit optimization is most effective when a large spectrum of mutation operations is available to the algorithm.

Over time, evolutionary approaches to circuit design have been investigated further, yielding methods that include recombination as an evolution operator [150, 151] or use more elaborate mutation schemes involving groups of gates [152], showing that the idea of *growing* problem-specific quantum circuits remains relevant.

6. Parameter Compression for Adiabatic Quantum Computing

So far we have seen that AQC in the form of QA can be used as part of a ML pipeline to perform Feature Selection. Generally, QA is a powerful and versatile tool for solving discrete optimization problems that arise throughout diverse application areas. In Chapters 7 and 8, we will investigate two more applications of QA in the form of Qubo formulations of two very different optimization problems. Particularly, in Chapter 7 a quantum annealer is itself used within an optimization loop. A so-far open question is whether the results produced by quantum annealers can actually compete with classical Qubo solvers, and which factors influence the solution quality in NISQ devices.

Although quantum computing holds the promise of speeding up classical computations, there has not yet been conclusive evidence that AQC is definitely faster than classical computing resources [153]. While some classical hardware-based Qubo solvers have been developed as alternatives to imperfect quantum devices [40, 41] to facilitate research and practical applications, a problem common to all hardware solvers is the limited parameter precision: In theory, Qubo is defined with real-valued weights (see Section 2.2.3), but digital devices utilize finite number representations, typically binary registers of length $B \in \{16, 32, 64\}$, which can represent 2^B distinct values. This necessarily leads to some loss of precision. The standard solution to this problem in classical CPUs and GPUs is floating-point arithmetic, as defined by, e.g., IEEE 754 [154].

A problem that is reminiscent of this scenario, but which is specific to D-Wave's quantum annealers, is Integrated Control Error (ICE) [155, 75], which randomly distorts the Ising model parameters, leading to a skewed energy landscape. Depending on the structure and parameters of the particular problem instance at hand, this distortion may be enough to change the optimum, as shown in Fig. 6.1: While a classical solver may find the correct minimizing solution (which may be intractable for large n), uploading the weight matrix to a hardware solver may lead to perturbations that result in a different minimizing solution. The solution found by the hardware device may be correct for the physical Qubo instance present on the device, but may not correspond to a solution of the original problem. As the weights on the QA device are analog, IEEE 754 or other types of floating point arithmetic are not applicable.

Not only quantum annealers, but also digital annealing hardware suffers from the effect of low parameter precision, due to rounding of problem parameters, e.g., when converting between floating point formats. There has been very little research investigating the effect

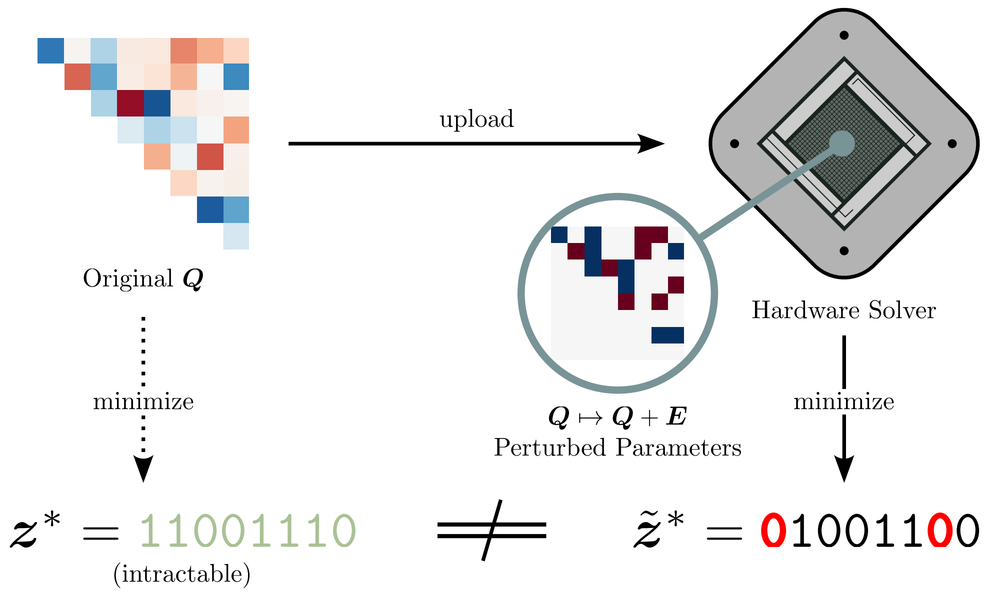

upload
Original $\boldsymbol{Q}$
minimize
Hardware Solver
minimize
$\boldsymbol{Q} \mapsto \boldsymbol{Q} + \boldsymbol{E}$
Perturbed Parameters
$\boldsymbol{z}^* = 11001110$
(intractable)
$\neq$
$\tilde{\boldsymbol{z}}^* = 01001100$

Figure 6.1.: QUBO solvers have a limited parameter resolution, leading to perturbations $\boldsymbol{E}$ that may result in false optima. Source: [1].

of rounding or distorting QUBO and Ising model parameters on their energy functions. However, we find that rounding can drastically reduce the probability to find the global optimum, depending on the parameter distribution: As Fig. 6.2 shows, a low number of parameter bits and a high DR (see Section 6.2 for details) leads to a higher probability of changing the minimizing vectors.

In this chapter, we investigate the implications of solving QUBO problems under parameter precision constraints. Over its course we will see that DR, a quantity proportional to the number of bits required to encode QUBO parameters faithfully, is a key factor for the solution quality under parameter distortions: A smaller DR implies that a QUBO instance is more robust against distortion. To formalize this notion, we use the optimum inclusion relation $\sqsubseteq$ introduced in Section 2.2.3, exploring the extent of parameter change under which this relation remains intact. Finally, we demonstrate techniques to reduce the DR of a given QUBO instance based on theoretical bounds on the optimal energy value, which we demonstrate experimentally. The results clearly show that the performance of QA hardware can be improved drastically following these strategies.

This chapter is based on the publication [5]. The author of this thesis developed the concept of QUBO parameter compression, devised and proved all theoretical insights, implemented and conducted most experiments and created their result plots, and wrote the majority of the article.

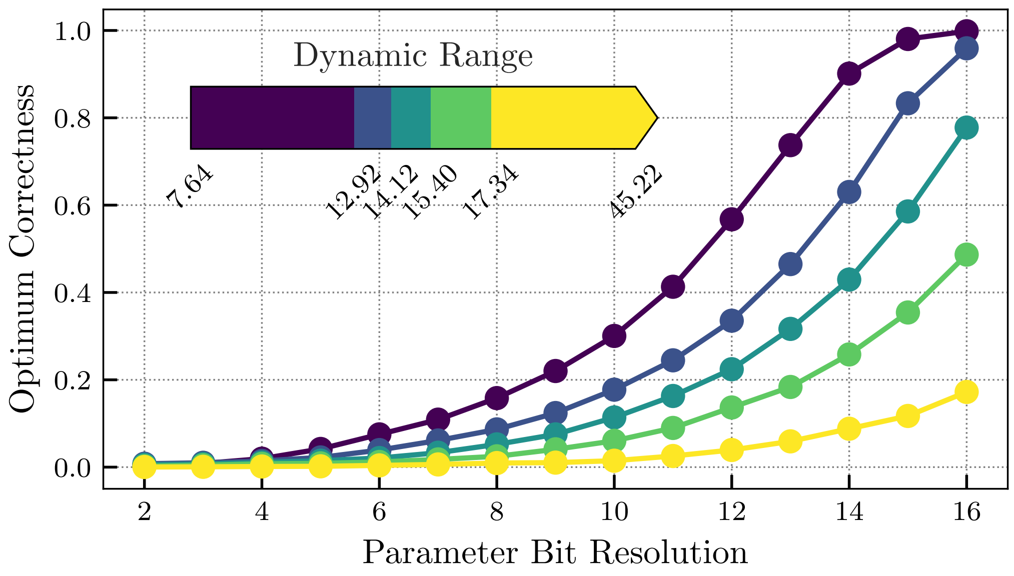


Figure 6.2.: The effect of rounding QUBO parameters to a certain number of bits on the probability that the optimum remains intact, using random instances of the SUBSETSUM problem with $n = 16$. Each line represents the mean taken over 20,000 instances from a bin containing problems with similar Dynamic Range (DR) (see Section 6.2). See Section 6.6.2 for details. Source: [1].

6.1. Preserving Optima

In this section we investigate how the precision of the QUBO weight representation (specifically after rounding) affects the energy landscape of $f_{\boldsymbol{Q}}$, and how we can evaluate the minimum number of bits necessary to represent the entries of $\boldsymbol{Q}$ faithfully. Using the concept of optimum inclusion introduced in Def. 2.7, we show that there is a theoretical lower bound on a scaling factor that preserves at least one of the optima after rounding the parameters.

Each instance of QUBO has at least one binary vector with minimal energy, which is global optimum of the optimization problem. As defined in Section 2.2.3, we denote the set of minimizing vectors of a QUBO instance $\boldsymbol{Q}$ by $S^*(\boldsymbol{Q}) \neq \emptyset$. As mentioned in Section 2.2.3, scaling the QUBO matrix with a positive factor α leaves the optima unchanged, i.e., $S^*(\boldsymbol{Q}) = S^*(\alpha\boldsymbol{Q})$. Now, the notion $\boldsymbol{Q} \sqsubseteq \boldsymbol{R}$ defined in Def. 2.7 expresses the intuition that all minimizing vectors of $\boldsymbol{Q}$ also minimize $\boldsymbol{R}$, as $S^*(\boldsymbol{Q}) \subseteq S^*(\boldsymbol{R})$. It formalizes the idea of *preserving* optima when making changes to the weight matrix.

Preserving *all* minimizing solutions of an optimization problem has some clear advantages: For one, we are left with a choice between equally good solutions and can pick one according to other criteria not covered by the loss function (e.g., lowest or highest numer of 1-bits). Moreover, when more optimal solutions are scattered across the search space, local search heuristics may converge to one of them with higher probability. Preserving only *some* solutions is a compromise, enabling us to compress the parameters more effectively while still leaving optimal solutions to the original problem untouched. As we have proven that the set of QUBO instances with only a single minimizing vector lies dense within $\mathcal{Q}_n$ in Proposition 2.2, i.e., most QUBO instances have only a single minimizing vector, maintaining the relation $\sqsubseteq$ will preserve all optima anyway in most practical scenarios.

6.2. The Role of Dynamic Range

To proceed, we need to establish a quantity that measures the degree of precision we need to faithfully represent QUBO weights. To this end, we define the Dynamic Range (DR) of QUBO instances, borrowing the general notion from signal processing (see, e.g., [156]).

Definition 6.1 (Dynamic Range [1])**.** *Let $X \subset \mathbb{R}$ be a finite set. First, we define the set of absolute differences between all elements as $D_{\text{set}}(X) = \{|x-y| : x, y \in X,\, x \neq y\}$, and we write $\check{D}_{\text{set}}(X) = \min D_{\text{set}}(X)$ and $\hat{D}_{\text{set}}(X) = \max D_{\text{set}}(X)$. The DR of X is defined as*

$$\mathrm{DR}_{\text{set}}(X) = \log_2\left(\frac{\hat{D}_{\text{set}}(X)}{\check{D}_{\text{set}}(X)}\right) \tag{6.1}$$

Its unit is bits*, and its domain is the positive real numbers, similar to entropy in information theory. The DR of a QUBO matrix $\boldsymbol{Q}$ is defined as the DR of the set of its entries:*

$$\mathrm{DR}(\boldsymbol{Q}) = \mathrm{DR}_{\text{set}}(\mathcal{U}(\boldsymbol{Q})), \qquad \textit{where } \mathcal{U}(\boldsymbol{Q}) = \{Q_{ij} : i, j \in \{1, \dots, n\}\}\,.$$

Note that always $0 \in \mathcal{U}(\boldsymbol{Q})$, because $\boldsymbol{Q}$ is triangular, thus $Q_{ij} = 0$ when $i > j$. Further we define $D(\boldsymbol{Q}) = D_{\text{set}}(\mathcal{U}(\boldsymbol{Q}))$, $\check{D}(\boldsymbol{Q}) = \check{D}_{\text{set}}(\mathcal{U}(\boldsymbol{Q}))$ and $\hat{D}(\boldsymbol{Q}) = \hat{D}_{\text{set}}(\mathcal{U}(\boldsymbol{Q}))$.

If the DR of an instance $\boldsymbol{Q}$ is large, it requires many bits to represent all weights faithfully in binary, as its parameters both stretch over a large value range while simultaneously requiring small gradations. Assum we use $k \in \mathbb{N}$ bits to represent unsigned integers. The value range is $0, 1, \dots, 2^k - 1$, resulting in a DR of $\log_2(2^k - 1)$, which is approximately equal to k, especially when k is large. For $k > 1$, the number of bits required to represent the weights can be obtained by rounding the DR up to the nearest integer. Note that positive scaling leaves DR unaffected, as both the value range as well as the gradations are scaled by the same amount.

Example 6.1 *(from [1]).* Consider the following QUBO parameter matrices $\boldsymbol{Q}, \boldsymbol{Q}' \in \mathcal{Q}_2$:

$$\boldsymbol{Q} = \begin{bmatrix} -1 & 14380 \\ 0 & -2 \end{bmatrix} \qquad \boldsymbol{Q}' = \begin{bmatrix} -1 & 3 \\ 0 & -2 \end{bmatrix}$$

Matrix $\boldsymbol{Q}$ defines the energy function $f_{\boldsymbol{Q}}(\boldsymbol{z}) = -z_1 + 14380 z_1 z_2 - 2z_2$. The global optimum is $\boldsymbol{z}^* = (0, 1)^\intercal$ with value $f_{\boldsymbol{Q}}(\boldsymbol{z}^*) = -2$. As we see at a glance, the value Q_{12}

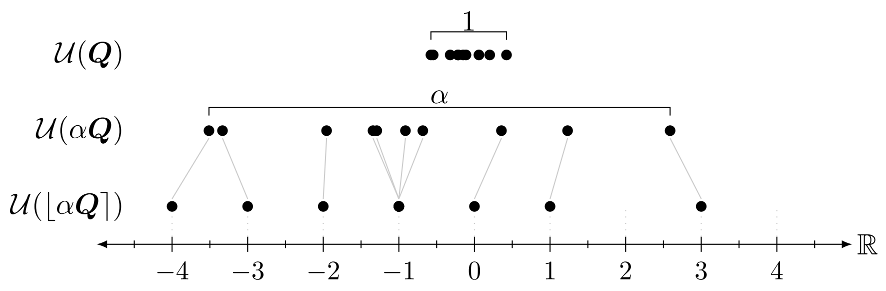

1
$\mathcal{U}(\boldsymbol{Q})$
α
$\mathcal{U}(\alpha\boldsymbol{Q})$
$\mathcal{U}(\lfloor\alpha\boldsymbol{Q}\rceil)$
$\mathbb{R}$
-4
-3
-2
-1
0
1
2
3
4

Figure 6.3.: Illustration of Proposition 6.1: W.l.o.g. let $\hat{D}(\boldsymbol{Q}) = 1$. By scaling with $\alpha > 0$ and rounding, the smallest distance $\check{D}(\lfloor\alpha\boldsymbol{Q})\rceil$ becomes 1, while the largest distance obeys $\hat{D}(\lfloor\alpha\boldsymbol{Q})\rceil < \alpha + 1$, as the maximal absolute rounding error is 0.5 for both outermost weights. Source: [1].

is positive and very large, acting as a penalty weight between bits z_1 and z_2. However, a much smaller value has the same effect, as we see with $\boldsymbol{Q}'$, which is identical to $\boldsymbol{Q}$ except for $Q'_{12} = 3$. The global optimum of $f_{\boldsymbol{Q}'}$ is still $\boldsymbol{z}^* = (0,1)^\intercal$ with $f_{\boldsymbol{Q}'}(\boldsymbol{z}^*) = -2$, therefore $\boldsymbol{Q} \sqsubseteq \boldsymbol{Q}'$. When we compare the DRs of $\boldsymbol{Q}$ and $\boldsymbol{Q}'$, we find that

$$\begin{aligned}\mathrm{DR}(\boldsymbol{Q}) &= \log_2\left(\frac{14380-(-2)}{0-(-1)}\right) & \qquad \mathrm{DR}(\boldsymbol{Q}') &= \log_2\left(\frac{3-(-2)}{0-(-1)}\right)\\ &\approx 13.812 & &\approx 2.322,\end{aligned}$$

which is a tremendous reduction: While we need 14 bits to encode the elements of $\boldsymbol{Q}$, we only need 3 for $\boldsymbol{Q}'$. This demonstrates that, in principle, it is possible to reduce the DR while preserving the minimizing vectors of the Qubo problem.

A straightforward way to enforce a reduction of DR is scaling and rounding the parameters, which sets the smallest meaningful difference between values to 1. It is easy to see that this gives us an upper bound on the DR:

Proposition 6.1 (Rounding bounds DR [1]). Let $\boldsymbol{Q} \in \mathcal{Q}_n$; w.l.o.g. we assume that $\boldsymbol{Q}$ is scaled such that $\hat{D}(\boldsymbol{Q}) = 1$. For any $\alpha > 0$ the DR of $\lfloor\alpha\boldsymbol{Q}\rceil$ is bounded above by

$$\mathrm{DR}(\lfloor\alpha\boldsymbol{Q}\rceil) < \log_2(\alpha+1)\ .$$

Proof. Due to normalization we know that $\hat{D}(\alpha\boldsymbol{Q}) = \alpha$. By rounding we enforce $\check{D}(\lfloor\alpha\boldsymbol{Q})\rceil \geq 1$, and $\hat{D}(\lfloor\alpha\boldsymbol{Q})\rceil = \alpha+\epsilon$ with rounding error $\epsilon = \hat{D}(\lfloor\alpha\boldsymbol{Q})\rceil - \alpha$. As $\hat{D}(\alpha\boldsymbol{Q})$ is the difference between the largest and smallest weights in $\alpha\boldsymbol{Q}$, and rounding introduces an error in $(-\frac{1}{2}, \frac{1}{2}]$ for each weight, we find that $\epsilon \in (-1,1)$, and therefore $\alpha - 1 < \hat{D}(\lfloor\alpha\boldsymbol{Q}\rceil) < \alpha + 1$. This yields $\mathrm{DR}(\lfloor\alpha\boldsymbol{Q}\rceil) \leq \log_2((\alpha+\epsilon)/1) < \log_2(\alpha+1)$. See Fig. 6.3 for a visualization of this proof. $\square$

Rounding an instance $\boldsymbol{Q}$, while being an effective way of reducing its DR, does generally *not* preserve $S^*(\boldsymbol{Q})$, as it skews the values of $f_{\boldsymbol{Q}}$. We can model this perturbation caused by rounding after scaling with some $\alpha > 0$ as

$$\begin{aligned} f_{\lfloor \alpha \boldsymbol{Q} \rceil}(\boldsymbol{z}) &= \boldsymbol{z}^\intercal \lfloor \alpha \boldsymbol{Q} \rceil \boldsymbol{z} \\ &= \boldsymbol{z}^\intercal \big(\alpha \boldsymbol{Q} + \boldsymbol{E}(\boldsymbol{Q}, \alpha)\big) \boldsymbol{z} \\ &= \alpha \boldsymbol{z}^\intercal \boldsymbol{Q} \boldsymbol{z} + \boldsymbol{z}^\intercal \boldsymbol{E}(\boldsymbol{Q}, \alpha) \boldsymbol{z} \\ &= \alpha f_{\boldsymbol{Q}}(\boldsymbol{z}) + f_{\boldsymbol{E}(\boldsymbol{Q},\alpha)}(\boldsymbol{z}) \; . \end{aligned}$$

Here, $\boldsymbol{E}(\boldsymbol{Q}, \alpha) = \lfloor \alpha \boldsymbol{Q} \rceil - \alpha \boldsymbol{Q} \in (-\frac{1}{2}, \frac{1}{2}]^{n \times n}$ denotes the matrix of differences between the real weights and their nearest integers in $\boldsymbol{Q}$ after scaling with $\alpha > 0$. Dividing by α after rounding restores the problem's original scaling, which results in an error on each entry in $\boldsymbol{Q}$ bounded in $(-\frac{1}{2\alpha}, \frac{1}{2\alpha}]$. As we can see, $\boldsymbol{E}(\boldsymbol{Q}, \alpha)$ itself is a QUBO weight matrix, therefore we can use its energy function as defined in Def. 2.5 and express the total error on the function value of any bit vector $\boldsymbol{z}$ as

$$\epsilon_{\boldsymbol{Q},\alpha}(\boldsymbol{z}) = f_{\boldsymbol{E}(\boldsymbol{Q},\alpha)}(\boldsymbol{z}) = \boldsymbol{z}^\intercal \boldsymbol{E}(\boldsymbol{Q}, \alpha) \boldsymbol{z} \; . \tag{6.2}$$

Representing the error caused by rounding in this way is meaningful, as it bridges the gap between rounding errors and ICE errors specific to D-Wave quantum annealers: The latter can be modeled by adding random noise to the underlying Ising model's parameters $\boldsymbol{J}$ and $\boldsymbol{h}$ [75], such that, in expectation, this type of error has the same form as Eq. (6.2).

6.3. An Optimal Rounding Strategy

Having found an effective way to reduce DR by force, the open question remains how we can ensure that all, or at least some, of the optima remain intact. Blindly rounding and checking if $\lfloor \alpha \boldsymbol{Q} \rceil \sqsubseteq \boldsymbol{Q}$ is intractable, as we proved in Proposition 2.1. However, as we see from Eq. (6.2), the rounding errors' magnitude relative to the weights' magnitude generally decreases as α increases, because $\forall i, j \in [n] : \; |\boldsymbol{E}(\boldsymbol{Q}, \alpha)_{ij}| \leq 0.5$, which is independent of $\boldsymbol{Q}$ and α. With this knowledge we can bound the error between $f_{\lfloor \alpha \boldsymbol{Q} \rceil / \alpha}$ and $f_{\boldsymbol{Q}}$ within an interval of $\pm C/\alpha$ for some constant $C > 0$. By choosing α such that the rounding errors cannot be larger than the gap between the lowest and second to lowest value of $f_{\boldsymbol{Q}}$, we can guarantee that no non-optimal vector's value is rounded down to the optima's value.

Definition 6.2 (Optimum Energy Gap [1]). *Let $\boldsymbol{Q} \in \mathcal{Q}_n$, and let*

$$\begin{aligned} y_1 &= f^*(\boldsymbol{Q}) = \min_{\boldsymbol{z} \in \mathbb{B}^n} f_{\boldsymbol{Q}}(\boldsymbol{z}) \\ y_2 &= \min_{\boldsymbol{z} \in \mathbb{B}^n \setminus S^*(\boldsymbol{Q})} f_{\boldsymbol{Q}}(\boldsymbol{z}) \end{aligned}$$

be the lowest and second to lowest energy values w.r.t. $f_{\boldsymbol{Q}}$, then the optimum energy gap *$\gamma_{\boldsymbol{Q}}$ is defined as*

$$\gamma_{\boldsymbol{Q}} = y_2 - y_1 \; .$$

This definition borrows the notion of the *spectral gap*, which is defined as the difference between the lowest and second to lowest eigenvalues of a Hamiltonian operator in physics, such as the Hamiltonian of the Ising model, and is difficult to quantify or compute [73]. The spectral gap of the Ising Hamiltonian is an important property that influences the performance of QA (cf. Section 2.4.3). However, we can extend this concept to any single-objective optimization problem with a real-valued loss function.

It is clear that computing the optimum energy gap is, in general, at least as hard as solving the QUBO problem itself, which is intractable for large n. However, it gives us a theoretical lower bound for α that allows for optimum-preserving rounding:

Theorem 6.1 (Optimum-Preserving Rounding [1]). Let $\boldsymbol{Q} \in \mathcal{Q}_n$ and $\gamma_{\boldsymbol{Q}}$ its optimum energy gap. Then

$$\forall \alpha \geq \alpha^* : \lfloor \alpha \boldsymbol{Q} \rceil \sqsubseteq \boldsymbol{Q} \qquad\qquad \text{where } \alpha^* = \frac{n^2+n}{4\gamma_{\boldsymbol{Q}}} \,.$$

Proof. Let $\boldsymbol{Q} \in \mathcal{Q}_n$ be arbitrary but fixed. Recall Eq. (6.2), and that each element of $\boldsymbol{E}(\boldsymbol{Q},\alpha) = \lfloor \alpha \boldsymbol{Q} \rceil - \alpha \boldsymbol{Q}$ is bounded in $(-\frac{1}{2}, \frac{1}{2}]$ for all $\alpha \in \mathbb{R}_+$. Let $\boldsymbol{z}' \in S^*(\boldsymbol{Q})$ and $\boldsymbol{z} = \arg\min_{\boldsymbol{z}'' \in \mathbb{B}^n \setminus S^*(\boldsymbol{Q})} f_{\boldsymbol{Q}}(\boldsymbol{z}'')$, i.e., vectors of second-to-lowest and lowest energy w.r.t. the $\boldsymbol{Q}$-instance. Then we find that for all $\alpha > 0$,

$$\begin{aligned} f_{\lfloor \alpha \boldsymbol{Q} \rceil}(\boldsymbol{z}) - f_{\lfloor \alpha \boldsymbol{Q} \rceil}(\boldsymbol{z}') &= (f_{\alpha \boldsymbol{Q}}(\boldsymbol{z}) + \epsilon_{\boldsymbol{Q},\alpha}(\boldsymbol{z})) - (f_{\alpha \boldsymbol{Q}}(\boldsymbol{z}') + \epsilon_{\boldsymbol{Q},\alpha}(\boldsymbol{z}')) \\ &= \underbrace{f_{\alpha \boldsymbol{Q}}(\boldsymbol{z}) - f_{\alpha \boldsymbol{Q}}(\boldsymbol{z}')}_{\gamma_{\alpha \boldsymbol{Q}}} + \epsilon_{\boldsymbol{Q},\alpha}(\boldsymbol{z}) - \epsilon_{\boldsymbol{Q},\alpha}(\boldsymbol{z}') \\ &= \gamma_{\alpha \boldsymbol{Q}} + \underbrace{\sum_{i=1}^{n} \sum_{j=i}^{n} \underbrace{\boldsymbol{E}(\boldsymbol{Q},\alpha)_{ij}}_{\in(-\frac{1}{2},\frac{1}{2}]} \underbrace{(z_i z_j - z'_i z'_j)}_{\in\{-1,0,1\}}}_{\in(-\frac{n^2+n}{4},\frac{n^2+n}{4}]} \,. \end{aligned}$$

This shows that the error on the optimum energy gap caused by scaling and rounding is in the interval $(-\frac{n^2+n}{4}, \frac{n^2+n}{4}]$. From Def. 6.2 it is clear that $\forall \alpha > 0 : \; \gamma_{\alpha \boldsymbol{Q}} = \alpha \gamma_{\boldsymbol{Q}}$. To ensure that the rounding error cannot surpass $\alpha \gamma_{\boldsymbol{Q}}$, we have to choose α such that $\alpha \gamma_{\boldsymbol{Q}} > \frac{n^2+n}{4}$, which implies $\alpha > \frac{n^2+n}{4\gamma_{\boldsymbol{Q}}}$. Defining $\alpha^* = \frac{n^2+n}{4\gamma_{\boldsymbol{Q}}}$ concludes the proof. □

As this bound assumes many worst cases, it is rather loose and depends on factors such as the sparsity of $\boldsymbol{Q}$, the Hamming distance between $\boldsymbol{z}$ and $\boldsymbol{z}'$, and the specific non-zero values of $\boldsymbol{Q}$. Some simple refinements can be made: E.g., if we know that $\boldsymbol{Q}$ has exactly K non-zero entries, the bound can be made sharper by setting $\alpha^* = K/(2\gamma_{\boldsymbol{Q}})$. However, it ultimately depends on $\gamma_{\boldsymbol{Q}}$, which we cannot compute in practice. As the value of α^* depends inversely on $\gamma_{\boldsymbol{Q}}$, the proof still holds for any lower bound of $\gamma_{\boldsymbol{Q}}$.

6.4. Qubo Parameter Compression

So far we have seen that scaling and rounding reduces the DR of a Qubo instance reliably. However, ensuring that its optima are preserved, we need access to its optimum energy gap, which is intractable to compute. This leaves us, up to this point, with only a theoretical strategy to compress parameters. In this section we develop practical strategies to reduce the DR while keeping (a subset of) optimal vectors intact. To this end, instead of rounding all parameters at once, we modify single parameter values while trying to stay within bounds guaranteeing that a minimal solution stays minimal, and a non-minimal solution does not become minimal.

Justified by Proposition 2.2, we assume for now that a Qubo instance $\boldsymbol{Q} \in \mathcal{Q}_n$ has a unique global minimizer $\boldsymbol{z}^*$ of value $y^* = f_{\boldsymbol{Q}}(\boldsymbol{z}^*)$. The two competing objectives of *(i)* reduing the DR and *(ii)* maintaining the optimum inclusion relation can be summarized as a constrained optimization problem, where we want to find a matrix $\boldsymbol{A}^* \in \mathcal{Q}_n$ such that

$$\begin{aligned} \boldsymbol{A}^* = \operatorname*{arg\,min}_{\boldsymbol{A} \in \mathcal{Q}_n} \quad & \mathrm{DR}(\boldsymbol{Q}+\boldsymbol{A}) \\ \text{s.t.} \quad & \boldsymbol{Q}+\boldsymbol{A} \sqsubseteq \boldsymbol{Q}. \end{aligned} \tag{6.3}$$

The scaling and rounding strategy simply sets $\boldsymbol{A}^* = \boldsymbol{E}(\boldsymbol{Q}, \alpha^*)$, which is, however, by no means guaranteed to be the globally optimal solution of Eq. (6.3) (in fact, it is most definitely *not* in the majority of cases). To better approach this problem step by step, we update the elements of $\boldsymbol{Q}$ sequentially, setting $Q_{kl} \mapsto Q_{kl} + w_{kl}$ for a series of index pairs $k, l \in \{1, \dots, n\}$ with $k \leq l$. The general idea of iterative improvement by applying small changes is adjacent to the approach in Chapter 5, where quantum circuit are modified in a similar fashion.

In the following sections we will discuss how to choose the value w_{kl} to update the corresponding Qubo weight without changing the optimum. Later, we will see how to choose the indices k and l, but for now, accusem that $k \leq l$ are arbitrary but fixed.

6.4.1. Bounding the Minimal Energy

Recall the notion of binary vector subspaces $\mathbb{B}^n_{I \leftarrow \boldsymbol{z}}$ defined in Def. 2.4. If we fix one or more bits at indices I in a binary vector to constants, we implicitly define subspaces of $\mathbb{B}^n$, one for each possible assignment of variables indexed by I, i.e., $2^{|I|}$ in total. Each subspace has its own set of minimizing binary vectors w.r.t. $f_{\boldsymbol{Q}}$.

Definition 6.3 (Subspace Optima [1]). *Given $\boldsymbol{Q} \in \mathcal{Q}_n$, indices $I \subseteq \{1, \dots, n\}$ and an assignment $\boldsymbol{z} \in \mathbb{B}^{|I|}$, the subspace optima are defined as*

$$S^*_{I \leftarrow \boldsymbol{z}}(\boldsymbol{Q}) = \left\{ \boldsymbol{s}^* : \, \boldsymbol{s}^* \in \mathbb{B}^n_{I \leftarrow \boldsymbol{z}}, \, f_{\boldsymbol{Q}}(\boldsymbol{s}^*) \leq f_{\boldsymbol{Q}}(\boldsymbol{s}) \, \forall \boldsymbol{s} \in \mathbb{B}^n_{I \leftarrow \boldsymbol{z}} \right\} .$$

Note that $S^*_{I \leftarrow \boldsymbol{z}}(\boldsymbol{Q}) \cap S^*_{I \leftarrow \boldsymbol{z}'}(\boldsymbol{Q}) = \emptyset$ for any $\boldsymbol{z}, \boldsymbol{z}' \in \mathbb{B}^n$ with $\boldsymbol{z} \neq \boldsymbol{z}'$, as subspaces are always disjoint. For this reason, we can choose an arbitrary but fixed element $\boldsymbol{z}^*_{ab} \in$

$S^*_{kl\leftarrow ab}(\boldsymbol{Q})$ as a representative for all $(a,b) \in \mathbb{B}^2$, whose value we denote by $y^*_{ab} = f_{\boldsymbol{Q}}(\boldsymbol{z}^*_{ab})$. For brevity, we write $\mathbb{B}^n_{kl\leftarrow ab}$ instead of $\mathbb{B}^n_{\{i,j\}\leftarrow(a,b)^\intercal}$. Further, we write ab instead of $kl \leftarrow ab$ in the index of vectors and scalars, making k and l implicit from now on.

Naturally, the minimal values y^*_{ab} are just as hard to compute as solving the original QUBO itself. Instead of computing the exact values, we assume that we have upper and lower bounds for them, which are much easier to compute, and which we consequently to determine the update parameter w_{kl}.

Definition 6.4 (Subspace Minimal Energy Bounds [1]). *Upper (lower) bounds for y^*_{ab} are denoted by $\hat{y}_{ab}$ ($\check{y}_{ab}$), such that*

$$\check{y}_{ab} \leq y^*_{ab} \leq \hat{y}_{ab},\ \forall (a,b) \in \mathbb{B}^2 \ .$$

Further, let

$$\begin{aligned} y^-_{kl} &= \min\{0, \min\{\hat{y}_{00}, \hat{y}_{01}, \hat{y}_{10}\} - \check{y}_{11}\}\ , \\ y^+_{kl} &= \max\{0, \min\{\check{y}_{00}, \check{y}_{01}, \check{y}_{10}\} - \hat{y}_{11}\}\ , \end{aligned}$$

if $k \neq l$. Otherwise, when $k = l$, let $y^-_{kl} = \min\{0, \hat{y}_{00} - \check{y}_{11}\}$ and $y^+_{kl} = \max\{0, \check{y}_{00} - \hat{y}_{11}\}$.

Theorem 6.2 (Optimum-Preserving Weight Update [1]). Let $\boldsymbol{Q} \in \mathcal{Q}_n$, and assume we perform an update $Q_{kl} \mapsto Q_{kl} + w_{kl}$. Given y^-_{kl} and y^+_{kl} as defined in Def. 6.4, then an optimum is preserved as long as

$$y^-_{kl} \leq w_{kl} \leq y^+_{kl}\ . \tag{6.4}$$

Proof. We focus on the case $k \neq l$, the case $k = l$ is analogous. The global minimum energy y^* must be equal to exactly one of the four subspaces' minimum energies $y^*_{00}, y^*_{01}, y^*_{10}, y^*_{11}$. Notice that changing Q_{kl} by w_{kl} affects only y^*_{11}. Assume $\boldsymbol{z}^* \neq \boldsymbol{z}^*_{11}$, then an optimum is preserved if

$$\begin{aligned} & y^*_{11} + w_{kl} \geq y^* \\ \Leftrightarrow\ & y^*_{11} + w_{kl} \geq \min\{y^*_{00}, y^*_{01}, y^*_{10}\} \\ \Leftarrow\ & \check{y}_{11} + w_{kl} \geq \min\{\hat{y}_{00}, \hat{y}_{01}, \hat{y}_{10}\}\ . \end{aligned} \tag{6.5}$$

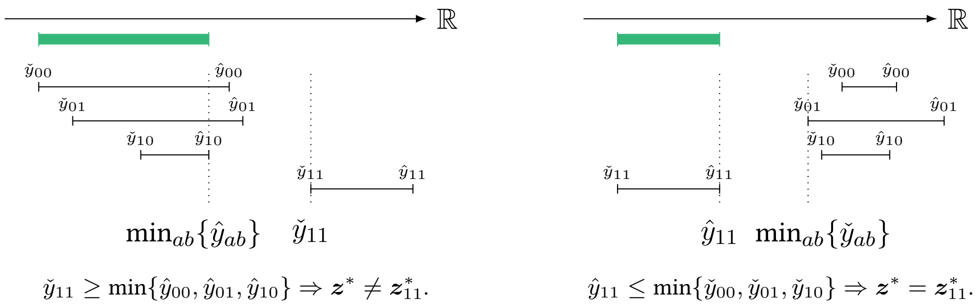
Furthermore, w_{kl} can take any positive value, since $y^*_{11} > y^*$. Combining this observation with Eq. (6.5), we end up with the lower bound

$$w_{kl} \geq \min\{0, \min\{\hat{y}_{00}, \hat{y}_{01}, \hat{y}_{10}\} - \check{y}_{11}\} = y^-_{kl}\ . \tag{6.6}$$

If $\boldsymbol{z}^* = \boldsymbol{z}^*_{11}$, we can similarly deduce an upper bound

$$w_{kl} \leq \max\{0, \min\{\check{y}_{00}, \check{y}_{01}, \check{y}_{10}\} - \hat{y}_{11}\} = y^+_{kl}\ . \tag{6.7}$$

Combining Eqs. (6.6) and (6.7) we obtain Eq. (6.4). □



$\mathbb{R}$
$\check{y}_{00}$
$\hat{y}_{00}$
$\check{y}_{01}$
$\hat{y}_{01}$
$\check{y}_{10}$
$\hat{y}_{10}$
$\check{y}_{11}$
$\hat{y}_{11}$
$\min_{ab}\{\hat{y}_{ab}\}$
$\check{y}_{11}$
$\check{y}_{11} \geq \min\{\hat{y}_{00}, \hat{y}_{01}, \hat{y}_{10}\} \Rightarrow \boldsymbol{z}^* \neq \boldsymbol{z}^*_{11}$.
$\hat{y}_{11}$
$\min_{ab}\{\check{y}_{ab}\}$
$\hat{y}_{11} \leq \min\{\check{y}_{00}, \check{y}_{01}, \check{y}_{10}\} \Rightarrow \boldsymbol{z}^* = \boldsymbol{z}^*_{11}$.

Figure 6.4.: Visual proof of Proposition 6.2 for $ab = (1, 1)$: The green bars represent the interval the global optimum must fall into. **Left:** When the lower bound for a subspace $\mathbb{B}^n_{kl\leftarrow ab}$ is greater than an upper bound of any other subspace, we can conclude that $\boldsymbol{z}^*_{ab}$ cannot be optimal. **Right:** When an the upper bound for a subspace $\mathbb{B}^n_{kl\leftarrow ab}$ is lower than the lower bounds of all other subspaces, we can conclude that $\boldsymbol{z}^*_{ab}$ is optimal. Source: [1].

Equation (6.4) uses bounds ($\check{y}_{ab}$ and $\hat{y}_{ab}$) on the true optima y^*_{ab} to define an interval. If we choose w_{kl} within this interval, the optimum is preserved. Under certain conditions, these bounds can also be used for determining optimality of $\boldsymbol{z}^*_{ab}$.

Proposition 6.2 (Optimality Conditions [1]). The following implications hold:

$$\check{y}_{ab} > \min\left(\{\hat{y}_{00}, \hat{y}_{01}, \hat{y}_{10}, \hat{y}_{11}\} \setminus \{\hat{y}_{ab}\}\right) \Rightarrow \boldsymbol{z}^* \neq \boldsymbol{z}^*_{ab} \tag{6.8}$$
$$\hat{y}_{ab} < \min\left(\{\check{y}_{00}, \check{y}_{01}, \check{y}_{10}, \check{y}_{11}\} \setminus \{\check{y}_{ab}\}\right) \Rightarrow \boldsymbol{z}^* = \boldsymbol{z}^*_{ab} \tag{6.9}$$

Proof. We focus on the case $k \neq l$, the case $k = l$ is analogous. Assume that Eq. (6.8) holds, i.e.,

$$\begin{aligned} &\check{y}_{ab} > \min\left(\{\hat{y}_{00}, \hat{y}_{01}, \hat{y}_{10}, \hat{y}_{11}\} \setminus \{\hat{y}_{ab}\}\right) \\ \Rightarrow\, &y^*_{ab} > \min\left(\{y^*_{00}, y^*_{01}, y^*_{10}, y^*_{11}\} \setminus \{y^*_{ab}\}\right) \\ \Leftrightarrow\, &y^*_{ab} > y^* \;\Leftrightarrow\; \boldsymbol{z}^* \neq \boldsymbol{z}^*_{ab}\ . \end{aligned}$$

The result in Eq. (6.9) follows analogously. See Fig. 6.4 for a visualization. □

If Eq. (6.9) is fulfilled for some two-bit assignment ab, we can fix $z_k = a$ and $z_l = b$ and reduce the Qubo size. This is very similar to the concept of *strong persistence* discussed in [157, 158, 92]. Knowing $\boldsymbol{z}^* \neq \boldsymbol{z}^*_{11}$, we can discard the upper bound Eq. (6.4) (cf. proof of Theorem 6.2).

After establishing the implications we can derive from lower and upper bounds on minimal energies in subspaces y^*_{ab}, the questions remains how to actually compute them. Every particular energy value of a Qubo instance $\boldsymbol{Q}$ is an upper bound on the minimal energy. Therefore, weak upper bounds can be computed very easily by evaluating the zero vector within the subspace:

Proposition 6.3 (Simple Upper Bound [1]). Let k, l be two indices with $1 \leq k \leq l \leq n$ and $(a, b) \in \mathbb{B}^2$ a variable assignment. An upper bound for y^*_{ab} is given by

$$y^*_{ab} \leq f_{\boldsymbol{Q}}(a\boldsymbol{e}_k + b\boldsymbol{e}_l).$$

Tighter upper bounds can be found by investing more computational effort, e.g., by performing a random or local search within $\mathbb{B}^n_{kl \leftarrow ab}$, and recording the lowest observed energy value.

To derive simple lower bounds we can take only the negative entries of $\boldsymbol{Q}$ and compute the lowest possible sum that can be formed from them.

Proposition 6.4 (Simple Lower Bound [1]). Let k, l and (a, b) as before. Define $\boldsymbol{Q}^-$ such that $Q^-_{ij} = \min\{0, Q_{ij}\}$, i.e., the matrix containing only the negative values of $\boldsymbol{Q}$. Then a lower bound for y^*_{ab} is given by

$$y^*_{ab} \geq f_{\boldsymbol{Q}^-}(\boldsymbol{1}_n + (a-1)\boldsymbol{e}_k + (b-1)\boldsymbol{e}_l).$$

We can find a significantly tighter bound by exploiting roof duality [159, 160], which derives lower bounds by determining the maximum flow of a corresponding flow network. The complexity of this algorithm is polynomial.

6.4.2. Reducing the Dynamic Range

Now that we have defined intervals wherin a single weight of a QUBO instance can be chosen without changing the optimum, we turn to the main objective of Eq. (6.3), i.e., we need to compute weight updates that actually reduce the DR. In the following we present multiple strategies to achieve this goal.

For convenience, we define $m = n^2$ as the number of entries of an $n \times n$ square matrix. The real-valued elements of $\boldsymbol{Q} \in \mathcal{Q}_n$ can be put in ascending order, giving rise to a function $\pi : \{1, \dots, m\} \to \{1, \dots, n\}^2$ with the defining property $Q_{\pi(i)} \leq Q_{\pi(i+1)}$ for all $1 \leq i < m$. As a shorthand, we define $q_i = Q_{\pi(i)}\ \forall 1 \leq i \leq m$. This allows us to write some of our previously defined quantities more concisely, e.g.,

$$\begin{aligned}
\min \mathcal{U}(\boldsymbol{Q}) &= q_1, \\
\max \mathcal{U}(\boldsymbol{Q}) &= q_m, \\
\hat{D}(\boldsymbol{Q}) &= q_m - q_1, \\
\check{D}(\boldsymbol{Q}) &= \min\{q_{i+1} - q_i : 1 \leq i < m\} \setminus \{0\}.
\end{aligned}$$

Note that, as $\boldsymbol{Q}$ is upper triangular, about half of all q_i are 0.

Theorem 6.3 (DR-Reducing Weight Update [1]). Let $\boldsymbol{Q} \in \mathcal{Q}_n$ and $\pi(\ell) = (k, l)$ with $k \leq l$ and $Q_{kl} \neq 0$. When adding a value $w_{kl} \in \mathbb{R}$ to Q_{kl}, the DR does not increase, i.e., $\mathrm{DR}(\boldsymbol{Q}) \geq \mathrm{DR}(\boldsymbol{Q} + w_{kl}\boldsymbol{e}_k\boldsymbol{e}_l^\top)$, if the following two conditions hold:

1. w_{kl} is bounded by

$$\underbrace{q_1 - q_\ell + \mathbb{1}\{\ell = m\}\,(q_{m-1} - q_m) - D^*_\ell}_{=:d^-_\ell} \le w_{kl} \le \underbrace{q_m - q_\ell + \mathbb{1}\{\ell = 1\}\,(q_2 - q_1) + D^*_\ell}_{=:d^+_\ell}, \tag{6.10}$$

2. w_{kl} does not decrease the minimal parameter distance:

$$|q_\ell + w_{kl} - q_i| \ge \check{D}(\boldsymbol{Q}), \qquad \forall i \in \{1, \dots, m\} \setminus \{\ell\} \tag{6.11}$$

$$\vee \qquad q_\ell + w_{kl} \in \mathcal{U}(\boldsymbol{Q}), \tag{6.12}$$

where D^*_ℓ is defined as

$$D^*_\ell = \hat{D}(\boldsymbol{Q}) \left(\frac{\check{D}_{\text{set}}(\{q_u :\ u \neq \ell\})}{\check{D}(\boldsymbol{Q})} - 1 \right).$$

Proof. We only consider an increase of the QUBO parameter Q_{kl}, i.e. $w_{kl} > 0$, since the results for decreasing Q_{kl} can be deduced analogously. Firstly, consider the parameters $q_\ell > q_1$. If

$$w_{kl} \le q_m - q_\ell\ , \tag{6.13}$$

then $\hat{D}(\boldsymbol{Q})$ is not increased and thus to avoid an increase of the DR, $\check{D}(\boldsymbol{Q})$ should not be decreased (see Eq. (6.1)). This can be achieved by maintaining a distance of at least $\check{D}(\boldsymbol{Q})$ to all other QUBO parameters, i.e.,

$$|q_\ell + w_{kl} - q_i| \ge \check{D}(\boldsymbol{Q}),\ \forall i \neq \ell, \tag{6.14}$$

or assuming the exact value of an already existing QUBO parameter, i.e.,

$$q_\ell + w_{kl} \in \mathcal{U}(\boldsymbol{Q}). \tag{6.15}$$

If the current maximum value is overshot, i.e., $w_{kl} > q_m - q_\ell$, then $\hat{D}(\boldsymbol{Q})$ is increased, and in order to reduce the DR, $\check{D}(\boldsymbol{Q})$ has to be increased as well. This can only happen if q_ℓ is unique and part of the minimum distance, i.e., $\check{D}_{\text{set}}(\{q_1, \dots, q_{\ell-1}, q_{\ell+1}, \dots, q_m\}) > \check{D}(\boldsymbol{Q})$. The change w_{kl} can then be bounded by

$$\begin{aligned}
& \mathrm{DR}\,(\boldsymbol{Q}) \ge \mathrm{DR}\left(\boldsymbol{Q} + w_{kl}\boldsymbol{e}_k\boldsymbol{e}_l^\intercal\right) \\
\Leftrightarrow\ & \frac{\hat{D}(\boldsymbol{Q})}{\check{D}(\boldsymbol{Q})} \ge \frac{\hat{D}(\boldsymbol{Q}) + q_\ell + w_{kl} - q_m}{\min\{\check{D}_{\text{set}}(\{q_u :\ u \neq \ell\}), \check{D}(\boldsymbol{Q}) + w_{kl}\}} \\
& \qquad\quad \ge \frac{\hat{D}(\boldsymbol{Q}) + q_\ell + w_{kl} - q_m}{\check{D}_{\text{set}}(\{q_u :\ u \neq \ell\})} \\
\Leftrightarrow\ & w_{kl} \le q_m - q_\ell + \hat{D}(\boldsymbol{Q}) \left(\frac{\check{D}_{\text{set}}(\{q_u :\ u \neq \ell\})}{\check{D}(\boldsymbol{Q})} - 1 \right).
\end{aligned}$$

Secondly, consider $\ell = 1$. If the smallest value q_1 is not unique, we can also deduce bounds Eqs. (6.13) to (6.15). On the other hand, when the smallest value is unique ($q_2 - q_1 > 0$) the increase $q_1 + w_{kl} > q_m$ does not necessarily increase $\hat{D}(\boldsymbol{Q})$. If $w_{kl} > q_2 - q_1$, $q_1 + w_{kl}$ is not the minimum value anymore but q_2 is. Thus, the difference $q_2 - q_1$ can be added to the bound in Eq. (6.13)

$$w_{kl} \leq (q_m - q_1) + (q_2 - q_1) = q_m - 2q_1 + q_2 \; .$$

Additionally, if q_1 is part of the unique minimum distance, we can add $q_2 - q_1$ to the bound in Eq. (6.16)

$$w_{kl} \leq q_m - q_1 + q_2 - q_1 + \hat{D}(\boldsymbol{Q}) \left(\frac{\check{D}_{\text{set}}(\{q_u : u \in [m] \setminus \{\ell\}\})}{\check{D}(\boldsymbol{Q})} - 1 \right) \; . \tag{6.16}$$

For a negative change $w_{kl} < 0$, similar bounds can be obtained analogously. □

Theorem 6.3, and Eqs. (6.10) to (6.12) in particular, provide loose bounds on feasible Qubo weight changes which we can safely apply. In the following sections, we discuss several heuristic approaches to choose specific values within these intervals.

6.5. Heuristic Compression Strategies

Our bounds on w_{kl} give us some freedom regarding the amount of change we want to allow for $\boldsymbol{Q}$. Firstly, we will look at a greedy strategy that exploits the interval fully to reduce the DR as much as possible. Afterwards, we will see how we can restrict our choice of w_{kl} further to maintain certain properties of $\boldsymbol{Q}$.

Greedy Strategy This strategy will be denoted by G. Let $(k, l) = \pi(\ell)$, as before. The parameter Q_{kl} is increased if $q_\ell < 0$, and decreased otherwise. For increasing (decreasing) Q_{kl} we choose w^{G}_{kl} maximally (minimally) within the interval given by Theorem 6.3, i.e., $w^{\text{G}}_{kl} = d^+_\ell$ ($w^{\text{G}}_{kl} = d^-_\ell$). However, if the updated parameter $Q_{kl} + w^{\text{G}}_{kl}$ is too close to another parameter and would cause $\check{D}(\boldsymbol{Q})$ to decrease, we set it equal to the next smaller (larger) parameter:

$$w^{\text{G}}_{kl} = \begin{cases} d^+_\ell & \text{if } q_\ell < 0 \wedge \min\{|q_\ell + d^+_\ell - q_i| : i \neq \ell\} \geq \check{D}(\boldsymbol{Q}), \\ \max\{q_j : q_j < q_\ell + d^+_\ell\} - q_\ell & \text{if } q_\ell < 0 \wedge \min\{|q_\ell + d^+_\ell - q_i| : i \neq \ell\} < \check{D}(\boldsymbol{Q}), \\ d^-_\ell & \text{if } q_\ell \geq 0 \wedge \min\{|q_\ell + d^-_\ell - q_i| : i \neq \ell\} \geq \check{D}(\boldsymbol{Q}), \\ \min\{q_j : q_j > q_\ell + d^-_\ell\} - q_\ell & \text{otherwise.} \end{cases} \tag{6.17}$$

Again, recall that there is always a $q_u = 0$ for some $u \in \{1, \dots, m\}$, therefore it is always possible to set parameters to 0. This is particularly useful for QA devices, because a parameter of 0 does not require a coupling between the qubits indexed by k and l, allowing for more flexibility in the assignment between logical and physical qubits. For this reason, we also introduce an alternative strategy G_0, which prefers the value 0 if it falls within the interval:

$$w_{kl}^{\mathsf{G}_0} = \begin{cases} -q_\ell & \text{if } q_\ell < 0 \wedge q_\ell + d_\ell^+ \geq 0, \\ -q_\ell & \text{if } q_\ell > 0 \wedge q_\ell + d_\ell^- \leq 0, \\ w_{kl}^{\mathsf{G}} & \text{otherwise.} \end{cases}$$

Maintaining the Parameter Ordering So far, we allowed the weights to cross over other weights, which changes their internal ordering. Another intuitive strategy to preserve the optimum is to maintain this ordering by *not* allowing weights to cross over neighboring weights. We call this strategy M. To this end, we define bounds on each QUBO weight q_ℓ as

$$\begin{aligned} \hat{q}_\ell^+ &= \min\{q_t :\ q_t > q_\ell\}, &(6.18)\\ \check{q}_\ell^+ &= \max\{q_t :\ q_t \leq q_\ell,\, t \neq \ell\}, \\ \hat{q}_\ell^- &= \min\{q_t :\ q_t \geq q_\ell,\, t \neq \ell\}, \\ \check{q}_\ell^- &= \max\{q_t :\ q_t < q_\ell,\}. &(6.19) \end{aligned}$$

Given that all entries of $\boldsymbol{Q}$ are unique, we find that $\hat{q}_\ell^+ = \hat{q}_\ell^-$ and $\check{q}_\ell^+ = \check{q}_\ell^-$, meaning that these bounds on q_ℓ differ only if $\boldsymbol{Q}$ contains duplicate weights. See Example 6.2 for an example of these bounds. Now, we want to update q_ℓ to lie exactly in the middle between $\check{q}_\ell^\pm$ and $\hat{q}_\ell^\pm$, which we denote by $\bar{q}_\ell^\pm$. If $q_1 < q_\ell < q_m$, we increase q_ℓ if $q_\ell - \check{q}_\ell^- < \hat{q}_\ell^+ - q_\ell$ and decrease otherwise, which gives us the update value

$$w_{kl}^{\mathsf{M}} = \begin{cases} \bar{q}_\ell^+ - \min\{\hat{q}_\ell^+ - q_\ell, q_\ell - \check{q}_\ell^+\}, & \text{if } q_\ell - \check{q}_\ell^- < \hat{q}_\ell^+ - q_\ell, \\ \bar{q}_\ell^- + \min\{\hat{q}_\ell^- - q_\ell, q_\ell - \check{q}_\ell^-\}, & \text{otherwise.} \end{cases}$$

We have to address the edge cases $\ell = 1$ and $\ell = m$ separately, as there is no midpoint as before. For $\ell = 1$ we define w_{kl}^{M} as

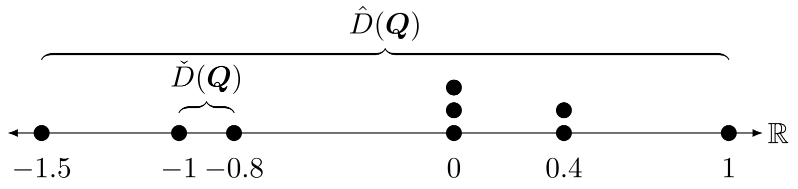
$$w_{kl}^{\mathsf{M}} = \begin{cases} \hat{q}_1^+ - q_1 - \check{D}(\boldsymbol{Q}), & \text{if } \check{D}_{\text{set}}(\{q_u :\ u > 1\}) = \check{D}(\boldsymbol{Q}), \\ \check{D}(\boldsymbol{Q}) - \check{D}_{\text{set}}(\{q_u :\ u > 1\}), & \text{otherwise.} \end{cases}$$

Analogously, for $\ell = m$ the weight update becomes

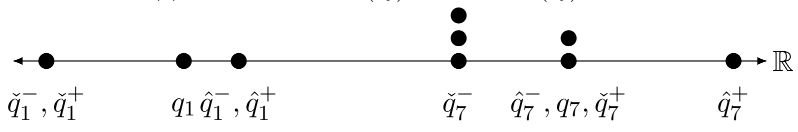
$$w_{kl}^{\mathsf{M}} = \begin{cases} \check{q}_m^- - q_m + \check{D}(\boldsymbol{Q}), & \text{if } \check{D}_{\text{set}}(\{q_u :\ u < m\}) = \check{D}(\boldsymbol{Q}), \\ \check{D}_{\text{set}}(\{q_u :\ u < m\}) - \check{D}(\boldsymbol{Q}), & \text{otherwise.} \end{cases}$$

Now that we have several strategies for computing weight updates to reduce the DR, we can determine the actual weight update w_{kl} by

$$w_{kl} = \min\{\max\{w_{kl}^h, y_{kl}^-\}, y_{kl}^+\}, \tag{6.20}$$



(a) We can read off $\hat{D}(\boldsymbol{Q}) = 2.5$ and $\check{D}(\boldsymbol{Q}) = 0.2$.


$\mathbb{R}$
$\check{q}_1^-, \check{q}_1^+$
$q_1\, \hat{q}_1^-, \hat{q}_1^+$
$\check{q}_7^-$
$\hat{q}_7^-, q_7, \check{q}_7^+$
$\hat{q}_7^+$

(b) Bounds (Eqs. (6.18) and (6.19)) on QUBO parameters $q_1 = -1$ and $q_7 = 0.4$.

Figure 6.5.: Sorted QUBO parameters of the matrix given in Eq. (6.21). Duplicates are indicated as vertically stacked points. Source: [1].

where $h \in \{\mathsf{G}, \mathsf{G}_0, \mathsf{M}\}$ is one of our heuristics. Clamping w_{kl} within $[y_{kl}^-, y_{kl}^+]$ ensures that our change does not fall out of the bounds that preserve the optimum. The following example illustrates how the method described in the following sections is applied to an actual QUBO instance.

Example 6.2 *(from [1]).* We define an arbitrary QUBO instance $\boldsymbol{Q} \in \mathcal{Q}_3$ as

$$\boldsymbol{Q} = \begin{bmatrix} -1 & 0.4 & 1 \\ 0 & 0.4 & -0.8 \\ 0 & 0 & -1.5 \end{bmatrix} . \tag{6.21}$$

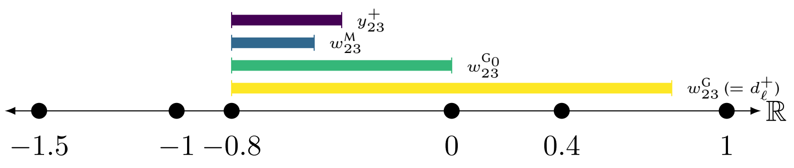
We can compute its DR to find $\mathrm{DR}(\boldsymbol{Q}) = \log_2(2.5/0.2) = 3.64$. If we put the parameters in order, we obtain

$$(q_1, \dots, q_9) = (-1.5, -1, -0.8, 0, 0, 0, 0.4, 0.4, 1) ,$$

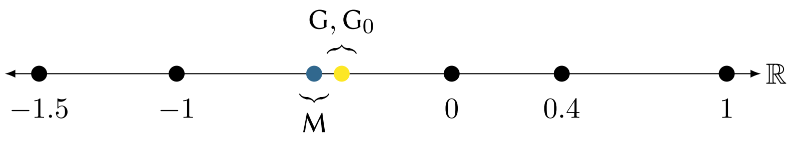
which are shown on the number line in Fig. 6.5. We show the bounds computed from Eqs. (6.18) and (6.19) for two exemplary weights in Fig. 6.5b. The value $Q_{23} = q_3 = -0.8$ should be increased, as this leads to a decrease of $\check{D}(\boldsymbol{Q})$ and, consequently, the DR. Therefore, we fix $k = 2$, $l = 3$. This small example allows us to easily see that $\boldsymbol{z}^* = (0, 1, 1)^\intercal$ is the minimizing binary vector of $\boldsymbol{Q}$. To maintain $\boldsymbol{z}^*$ as the optimum when changing Q_{23}, we need to stay within the bounds given by Theorem 6.2: Again, for this small example we compute the tightest possible bounds as

$$\check{y}_{00} = 0,\ \check{y}_{01} = -1.5,\ \check{y}_{10} = 0.4,\ \hat{y}_{11} = -1.9,$$

giving $y_{kl}^+ = \min\{0, -1.5, 0.4\} - 1.9 = 0.4$. This tells us that we can increase Q_{23} by at most 0.4 while maintining the optimality of $\boldsymbol{z}^*$, as seen in Fig. 6.6a. For our main objective of decreasing the DR we turn to the three heuristics G, G_0 and M, whose values come out as $w_{kl}^{\mathsf{M}} = 0.3$, $w_{kl}^{\mathsf{G}} = 1.6$ and $w_{kl}^{\mathsf{G}_0} = 0.8$, as shown in Fig. 6.6a. We find that M moves



(a) Parameter value intervals: Maximum increase $y^+_{23} = 0.4$ (blue) of QUBO parameter Q_{23} such that the optimum is preserved, along with heuristic interval limits $w^{\mathsf{M}}_{23} = 0.3$ (cyan), $w^{\mathsf{G}}_{23} = 1.6$ (green) and $w^{\mathsf{G}_0}_{23} = 0.8$ (yellow).



(b) New parameters after changing $Q_{23} = -0.8$ w.r.t. preserving the optimum: $\min\{y^+_{23}, w^{\mathsf{M}}_{23}\} = 0.3$ for M (cyan), and $\min\{y^+_{23}, w^{\mathsf{G}}_{23}\} = 0.4$ (yellow) for G (and G_0 analogously). In both cases, the DR decreases by one bit, since $\check{D}(\boldsymbol{Q})$ is doubled.

Figure 6.6.: Change of QUBO parameter Q_{23}. Source: [1].

Q_{23} to the midpoint of its neighbors. G increases Q_{23} as much as possible to reduce DR greedily as much as possible, while G_0 sets Q_{23} to 0. Figure 6.6b shows the final changes for all three heuristics. According to Eq. (6.20), the final weight update w_{23} is 0.3 for M and 0.4 for both G and G_0. All lead to $\check{D}(\boldsymbol{Q})$ becoming doubled, which decreases the DR to 2.64, saving one bit.

6.5.1. Choosing the Next Parameter

The last open question is how to decide which pair of indices (k, l) we should apply our aforementioned DR reduction strategy to. The simplest way would be to either *(i)* pick a random pair of indices, or *(ii)* iterate over all index pairs in sequence. Both of these methods become increasingly inefficient with growing n, as only a few different parameters directly determine the DR, namely those that directly define $\hat{D}(\boldsymbol{Q})$ and $\check{D}(\boldsymbol{Q})$ (cf. Fig. 6.5a), i.e., at most four. Changing any other parameter can only ever increase the DR, not decrease it. This realization leads to the much more efficient strategy of only choosing (k, l) among those index pairs whose parameters determine DR. From this point onward, we determine (k, l) by *(i)* computing the update values for all (up to) four index pairs, and *(ii)* greedily choose the one that leads to maximal DR reduction, choosing randomly whenever we encounter a tie.

6.6. Parameter Compression in Practice

We conduct a series of experiments that serve to demonstrate that the method described in the previous sections reduces the DR in practice. To this end, we the DR values of various QUBO instances *before* and *after* applying our method. Lastly, in Section 6.6.2 we

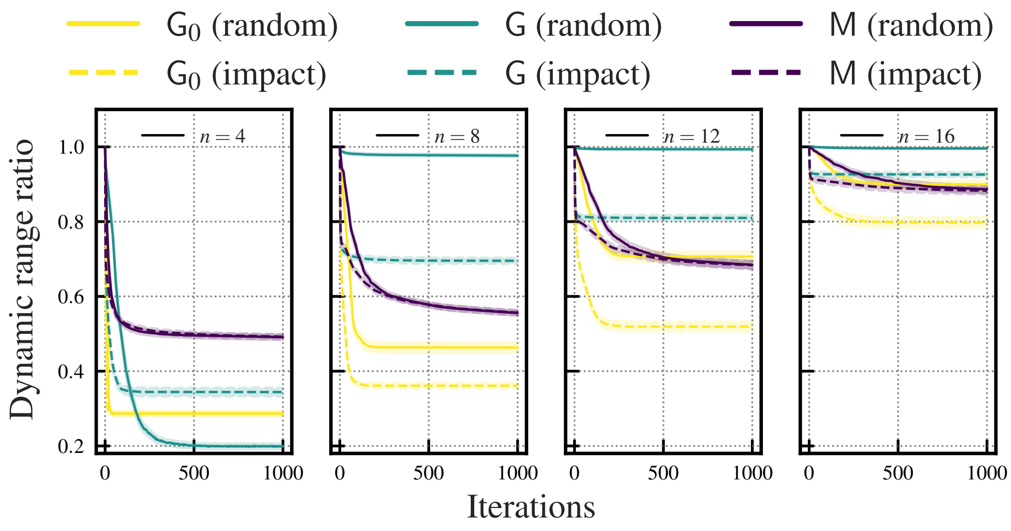


Figure 6.7.: DR ratio for different QUBO sizes $n \in \{4, 8, 12, 16\}$ over time. Source: [1].

compare the solutions obtained from a QA device using the *uncompressed* and *compressed* versions of the same QUBO weight matrix to show that a low DR does, in fact, improve solution quality.

6.6.1. Results for Random Instances

We try all compression heuristics discussed in Section 6.4.2, namely G (greedily choose the parameter update that leads to the greatest DR decrease), G_0 (like G, but prefer 0 values), and M (restrict bounds such that the parameter ordering remains intact), all of which are implemented as part of the Python package `qubolite`[19].

As a benchmark, we generate 1000 random QUBO instances each for $n \in \{4, 8, 12, 16\}$, whose weights we sample uniformly from the interval $[-0.5, 0.5]$. To each instance, we apply each heuristic given in Section 6.4.2 seperately for 1000 iterations. The upper bounds $\hat{y}_{ab}$ are found through a local search from a random initial binary vector. For the lower bounds $\check{y}_{ab}$ we use the roof-dual algorithm [159]. Additionally we compare the two methods described in Section 6.5.1, i.e., choosing the next QUBO weight randomly or greedily by its impact on the DR.

The results are shown in Fig. 6.7 as the DR ratio over time between the compressed and original QUBO matrices, that is $\mathrm{DR}(\boldsymbol{Q} + \boldsymbol{A}) / \mathrm{DR}(\boldsymbol{Q})$. The 95%-confidence intervals are indicated. As expected, the DR ratio decreases monotonically, showing that all heuristics do in fact reduce the DR. We observe that the decrease is more pronounced for smaller n, indicating that DR reduction gets gets more difficult the more parameters the instance

[19] `https://github.com/smuecke/qubolite` (last accessed June 3, 2025)

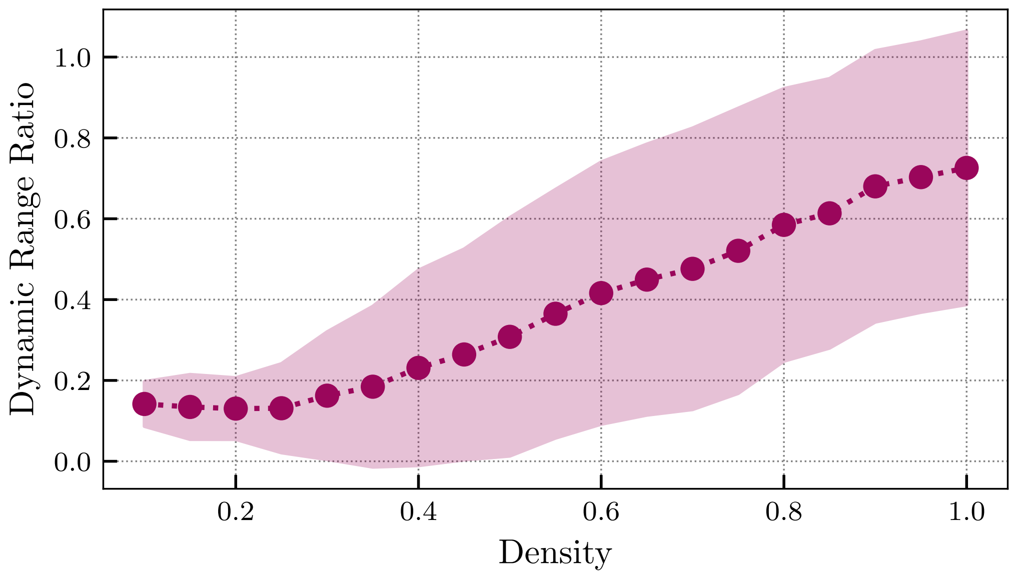


Figure 6.8.: DR ratio for random $\boldsymbol{Q}$-instances ($n = 16$) of varying densities. The mean DR ratios over 500 random instances per density value are shown as circles, their respective standard deviations as the area around them. Source: [1].

contains. This is also reflected in the parameter choice strategy, as choosing greedily by highest DR reduction leads almost always to a better overall reduction, except for $n = 4$. However, the greedy strategy comes at a higher computational cost, as all bounds on all relevant parameters have to be evaluated. For increasing n we further observe that M performs better than G, but worse than G$_0$, which shows that the strategy of setting weights to 0 when possible, reducing the overall number of unique parameter values in the process, turns out to be quite effective.

Lastly, we observe relatively quick convergence to a minimal DR ratio for all heuristics, with overall faster convergence for small n. As mentioned before, for larger n the minimum DR ratio approaches one, i.e., a smaller overall improvement is achieved. The iterative procedure most likely reaches a local optimum where no weight can be further modified without changing the minimizing vector. As the number of weights grows as $\mathcal{O}(n^2)$, this naturally occurs faster when n is small.

An implication that arises from this last observation is that DR reduction should be more effective when the QUBO weight matrix is sparse: To test this, we sample $\boldsymbol{Q}$-instances with $n = 16$ like before, but set each individual weight Q_{ij} to 0 with probability $1 - \rho$, where $\rho \in [0, 1]$ acts as a density parameter. This gives us weights matrices whose upper triangle has, in expectation, ρ non-zero parameters. For each value $\rho \in \{0.10, 0.15, 0.20, \dots, 1.00\}$ we sample 500 QUBO instances, apply to each 500 iterations of the G$_0$ DR reduction heuristic, and compute the final DR ratio. We choose fewer iterations here because of the quick convergence we observed in the previous experiments. The result is shown in

Fig. 6.8. Clearly, a low density (high sparsity) strongly correlates with a lower DR ratio, supporting our theory that the number of parameters is the decisive factor for the performance of DR reduction.

Using sharper bounds could improve performance further, but they require more computational effort. For instance, computing the roof dual bound has time complexity $\mathcal{O}(n^3)$. It is expected that clever implementations of the bounds could exploit the iterative nature of our DR reduction strategy, allowing for re-use and incremental updating of data structures (i.e., the flow network constructed for the roof dual bound), reducing the computation time drastically.

Impact on the Energy Landscape Some heuristic optimization methods behave differently when the ranking induced by the loss value of solution candidates changes due to DR reduction. For example, if for two distinct binary vectors $\boldsymbol{z}$ and $\boldsymbol{z}'$ we have $f_{\boldsymbol{Q}}(\boldsymbol{z}) < f_{\boldsymbol{Q}}(\boldsymbol{z}')$, but $f_{\tilde{\boldsymbol{Q}}}(\boldsymbol{z}) > f_{\tilde{\boldsymbol{Q}}}(\boldsymbol{z}')$ using the compressed instance $\tilde{\boldsymbol{Q}}$, an EA (see Section 2.2.2) would prefer $\boldsymbol{z}'$ over $\boldsymbol{z}$ using $\tilde{\boldsymbol{Q}}$, changing the convergence behavior. For this reason, we briefly investigate the effect of DR reduction on the ordering of binary vectors induced by the energy function.

To this end, we need to define the notion of the *induced ranking* of a QUBO instance, and a way to quantify the similarity of such rankings.

Definition 6.5 (Induced Ranking [1]). *Let $\boldsymbol{Q} \in \mathcal{Q}_n$, and let $\boldsymbol{z}^1, \boldsymbol{z}^2, \dots, \boldsymbol{z}^{2^n}$ be the binary vectors of $\mathbb{B}^n$ in lexicographical order. The* induced ranking *of $\boldsymbol{Q}$ is a permutation $\pi_{\boldsymbol{Q}} \in \{1, \dots, 2^n\} \to \{1, \dots, 2^n\}$ such that*

$$f_{\boldsymbol{Q}}(\boldsymbol{z}^{\pi_{\boldsymbol{Q}}(i)}) \leq f_{\boldsymbol{Q}}(\boldsymbol{z}^{\pi_{\boldsymbol{Q}}(i+1)})\ \forall i \in \{1, \dots, 2^n - 1\}\ .$$

Definition 6.6 (Kendall τ Distance [1]). *Let $\pi, \pi' : \{1, \dots, K\} \to \{1, \dots, K\}$ be two permutations for some $K > 1$. The normalized* Kendall τ distance *between π and π' is given by*

$$K_d(\pi, \pi') = \frac{1}{2} + \frac{1}{K(K-1)} \sum_{1 \leq i < j \leq K} \operatorname{sign}\left[(\pi(i) - \pi(j)) \cdot (\pi'(i) - \pi'(j))\right]\ .$$

Intuitively, the Kendall τ distance is the proportion of disagreement between rankings over all index pairs, i.e., the percentage of (i, j) such that $\pi(i) < \pi(j)$, but $\pi'(i) > \pi'(j)$, and vice versa. If $K_d(\pi, \pi') = 0$, then π and π' are identical, and if $K_d(\pi, \pi') = 1$, then π' is the exact reverse of π. For this reason we find that by computing $K_d(\pi_{\boldsymbol{Q}}, \pi_{\boldsymbol{Q}+\boldsymbol{A}})$ we obtain a useful measure of how strongly the DR reduction heuristic *scrambles* the energy landscape defined by $f_{\boldsymbol{Q}}$.

In Fig. 6.9 we compare the Kendall τ distance between the induced rankings of random QUBO instances before and after their DR reduction using all heuristics. We find that the compressed instances resulting from G and G$_0$ have to a much larger distance to their

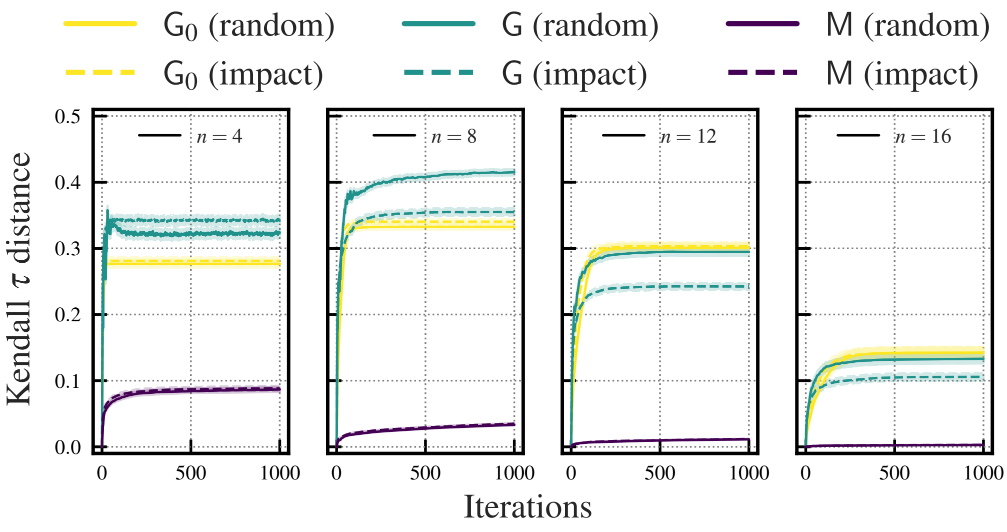


Figure 6.9.: State ordering for different QUBO sizes $n \in \{4, 8, 12, 16\}$. Source: [1].

original instances than those resulting from M. This was expected, as M preserves the internal ordering of parameters, which consequently leads to more conservative parameter changes, reducing the overall magnitude of *noise* added to the energy landscape.

Lastly, we analyze the number of unique parameter values of uncompressed and compressed QUBO instances. Figure 6.10 shows the unique weight ratio, i.e., the number of unique QUBO weights of the current iteration divided by the number of unique QUBO weights of the original QUBO. As expected, M does not allow weights to cross over their neighbors' values, thus never changing the number of unique weights. For G, the number of unique weights initially decreases, but then starts to increase again when more and more weights with non-unique values are changed to exploit the full range of their optimum-preserving intervals. On the other hand, using G^0, the ratio decreases seemingly monotonically, as the preference of 0 weights leads to fewer and fewer non-zero parameters.

6.6.2. Results on Quantum Hardware

In our last experiment, we want to quantify the extend that DR reduction can improve the performance of real quantum hardware. To this end, we *(i)* generate a QUBO instance $\boldsymbol{Q}$, *(ii)* apply DR reduction to obtain $\boldsymbol{Q}'$, *(iii)* apply QA to both $\boldsymbol{Q}$ and $\boldsymbol{Q}'$, performing multiple readouts, and *(iv)* compare the occurrence probabilities of the global minimizing vector.

Instead of using QUBO instances with randomly sampled weights, we use two exemplary real-world problems, BINCLUSTERING and SUBSETSUM. BINCLUSTERING stands for *binary*

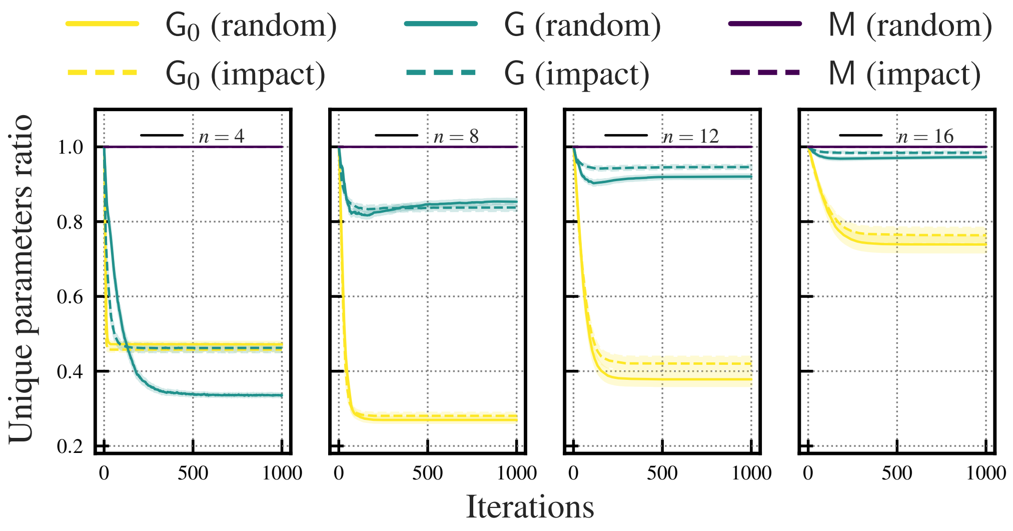


Figure 6.10.: Percentage of unique QUBO parameter values for different sizes $n \in \{4, 8, 12, 16\}$. Note that the line for method M (random) is constantly 1. Source: [1].

clustering and is an unsupervised ML task, where data points are assigned to one of two classes (or *clusters*) [9, Sec. 13.2.1]. SUBSETSUM is the problem of finding a subset from a list of values that sum up to a given target value. Both have well-established QUBO embeddings [38, 161]. We choose these as our example problems as they are *(i)* real-world problems of both scientific and economic interest, *(ii)* easy to generate for arbitrary n, and *(iii)* their corresponding QUBO instances' DR can be controlled by changing the input data accordingly. This last point in particular demonstrates that QUBO instances with a high DR can arise purely as a consequence of the original problem's input data, which we exploit to generate realistic high-DR QUBO instances (i.e., instances with a tangible connection to a real-world problem).

Generating a BINCLUSTERING Problem We generate input data for BINCLUSTERING by sampling i.i.d. $n = 20$ 2-dimensional points from an isotropic standard normal distribution, which we divide into two clusters by adding 4 to the first component of the first ten points, and by subtracting 4 from the first component of the latter ten points. This moves the points apart, creating two clusters. Lastly, we choose points 1 and 19 and multiply their coordinates by 20, which leads to a data set containing two outliers with a large magnitude (see Fig. 6.11).

From this data we derive a QUBO instance $\boldsymbol{Q}$ using the method from [38] using a linear kernel, which yields a standard 2-means clustering based on Euclidean distance. As a binary cluster assignment is symmetrical, we ensure that the QUBO problem has only one optimal solution by assigning class 0 to point 20, leaving only 19 points to optimize over. The QUBO weight matrix is shown in Fig. 6.12a. The color scale aptly illustrates the high

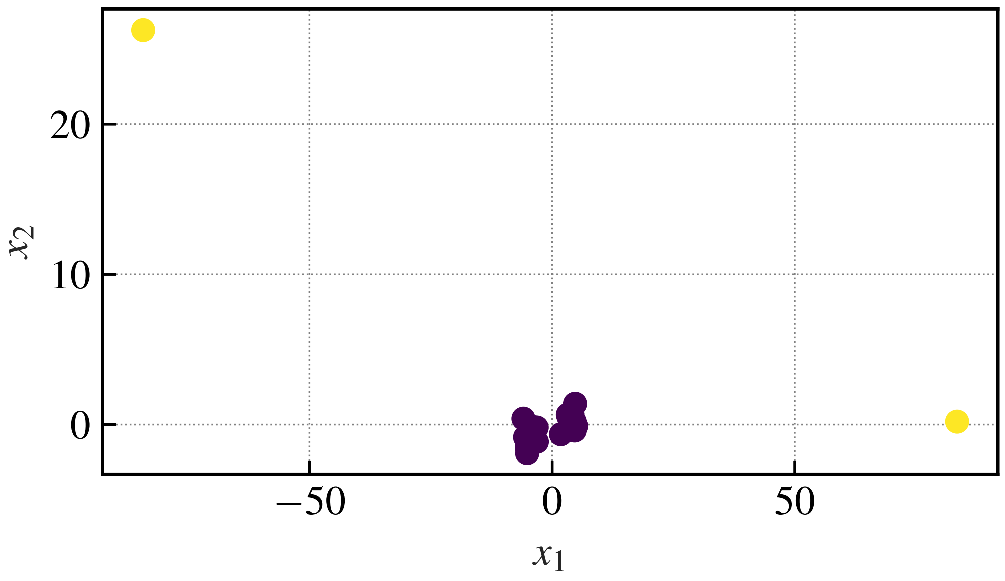


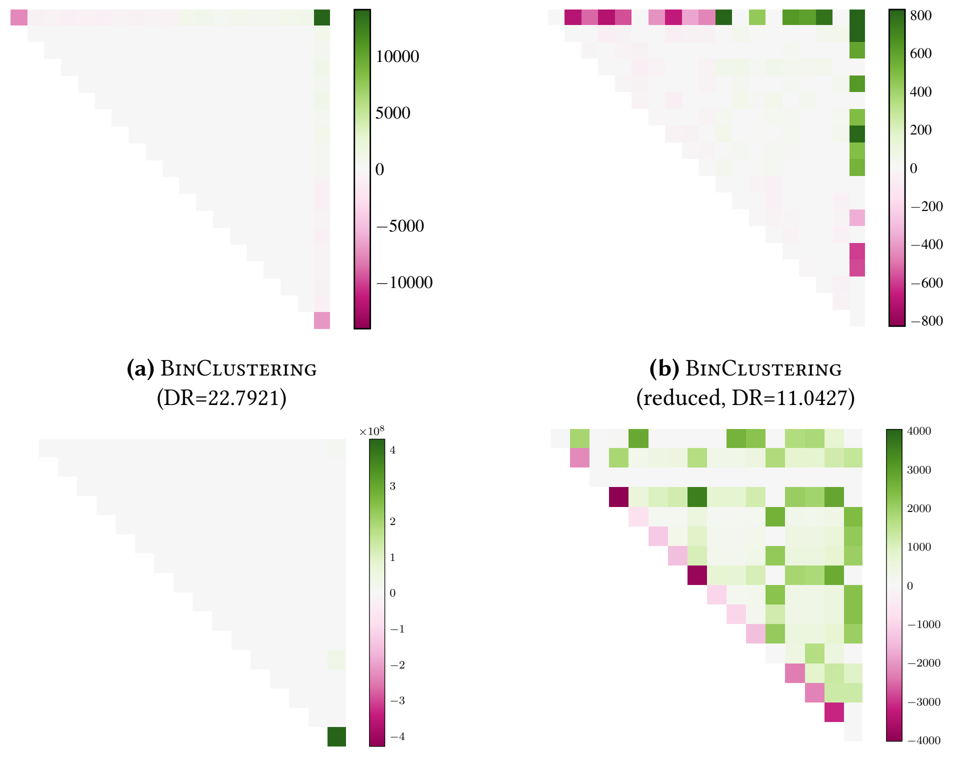
Figure 6.11.: 2D data set used for BinClustering, consisting of 20 points. The two outliers (shown in a different color) lead to large values in the Gram matrix, from which the Qubo parameters are derived, leading in turn to a large DR. Source: [1].

DR of 22.7921: Three weights have a very high magnitude while all others have a low magnitude, making them almost appear to have the same value.

We apply heuristic G_0 to $\boldsymbol{Q}$, since it seemed most promising in Section 6.6.1, having the additional benefit of making Qubo instances sparser. As before, we use a local search and the roof-dual algorithm for computing optimum-preserving bounds. To limit execution time, we give our algorithm a fixed budget of 100 iterations. The compressed Qubo instance $\boldsymbol{Q}'$ is shown in Fig. 6.12b. Clearly, much more detail is visible, as the color scale is much narrower.We find that $\mathrm{DR}(\boldsymbol{Q}') = 11.0427$, which is a reduction by more than half.

Generating a SubsetSum Problem In a similar way, we generate a SubsetSum problem, whose input data consists of a list $A = (a_1, \ldots, a_n)$ of n integers and a target value T. The task is to find a subset $S \subseteq \{1, \ldots, n\}$ such that $\sum_{i \in S} a_i = T$. This problem lends itself naturally to Qubo, where we use n binary variables which indicate if $i \in S$ for each i. The energy function is simply

$$f(\boldsymbol{x}) = \left(\sum_i a_i x_i - T \right)^2 \propto \sum_{i,j} a_i a_j x_i x_j - 2T \sum_i a_i x_i,$$



(a) BinClustering (DR=22.7921)

(b) BinClustering (reduced, DR=11.0427)

(c) SubsetSum (DR=25.6765)

(d) SubsetSum (reduced, DR=9.8882)

Figure 6.12.: Qubo parameter matrices for BinClustering and SubsetSum problems, before (left) and after (right) applying DR reduction. Difference in DR are illustrated by the color scale, rendering most parameter values in the original Qubo matrices indistinguishable, which is greatly improved on the right. Source: [1].

such that the Qubo parameters become

$$Q_{ij} = \begin{cases} 2a_i a_j & \text{if } i \neq j \\ a_i^2 - 2Ta_i & \text{otherwise.} \end{cases}$$

We set $n = 16$ and sample the elements of A i.i.d. as $|\lfloor 10 \cdot Z \rceil|$, where Z follows a standard Cauchy distribution[20]. This distribution is heavy-tailed, leading to occasional outliers with large magnitudes.Next, we sample the number of summands k from $\lfloor U \rceil$ where U follows a triangular distribution with parameters $a = \frac{n}{5}$, $b = \frac{n}{2}$ and $c = \frac{4n}{5}$, so that, on average, half of the elements of A contribute to the sum. Finally, we sample k indices from $\{1, \dots, n\}$ without replacement to obtain S, and set $T = \sum_{i \in S} a_i$. This sampling scheme was also used to generate Fig. 6.2. We use the same DR reduction method as before, but allow for 150 iterations. The weigt matrices are shown in Figs. 6.12c and 6.12d, which have a DR of 25.6765 and 9.8882 respectively. Again, the achieved compression

[20]If $X, Y \sim \mathcal{N}(0,1)$, then $Z = X/Y$ has standard Cauchy distribution (see, e.g., [162]).

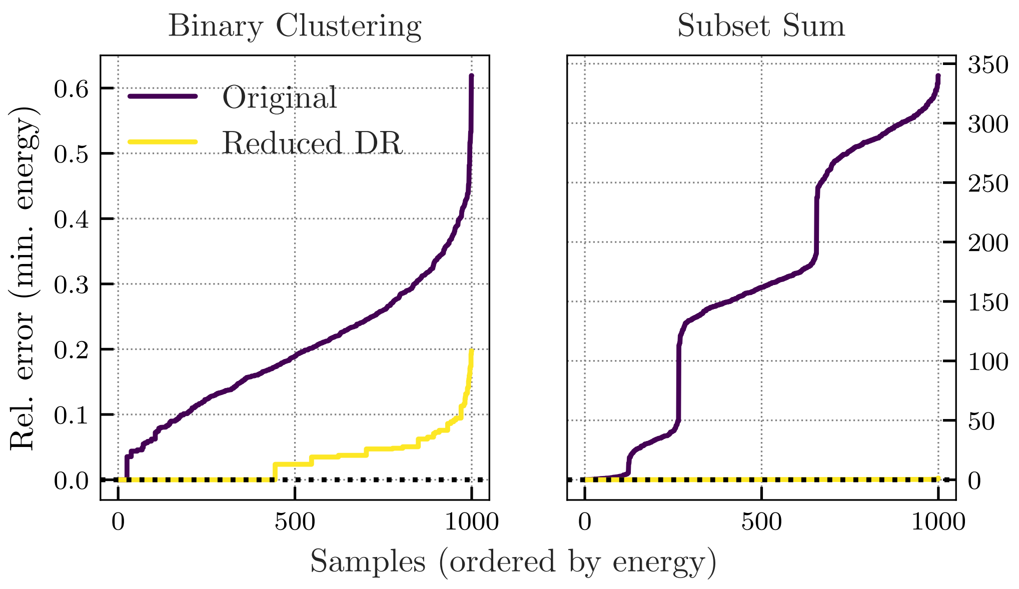


Figure 6.13.: Relative energies of 1000 samples obtained from the D-Wave quantum annealer for the original (dark blue) and compressed (yellow) QUBO instances. The samples are sorted in ascending order of energy. Lower is better. Source: [1].

reduces the number of required bits to encode this problem significantly, bringing the magnitude of the color scale from 10^8 down to 10^3.

We attempt to solve all four QUBO instances (both uncompressed and compressed for each of the two problems) on a D-Wave quantum annealer. To this end, we use the Advantage system 4.1, accessed through a Python interface. The D-Wave annealers have a fixed connectivity structure, i.e., only a subset of physical qubit pairs can be assigned a weight (cf. Section 2.4.3 under Limitations). Therefore, dense QUBO problems (as BINCLUSTERING and SUBSETSUM) must be embedded into this connectivity graph structure through redundant encoding and additional constraints. This is done automatically by D-Wave's Python package[21]. To eliminate the embedding as a contributing factor, we have this embedding computed once for the original QUBO and re-use it for the compressed one. This is possible because they have the same size, differing only in parameter magnitudes. In reality, one would indeed choose to re-compute the embedding, as the compressed instance is likely to be sparser and easier to embed.

For each QUBO instance, we perform 1000 readouts and record their energy values. The annealing time per read is $20\mu s$. Recall that the DR reduction method only keeps the solution vectors intact, but changes the specific energy values, which makes them incomparable to the original. For this reason, we compute the ground-truth minimal energies

[21]`https://docs.ocean.dwavesys.com/en/stable/concepts/embedding.html` (last accessed June 3, 2025)

v^* for both $\boldsymbol{Q}$ and $\boldsymbol{Q}'$, and plot the *relative deviation* of the measured energies v w.r.t. these values, $|(v^* - v)/v^*|$, along the y-axis in Fig. 6.13.

The figure clearly shows that the compressed QUBO instances yield the global optimum with much higher frequency than the original, uncompressed instances with high DR. We verified that the vector of minimal energy in the compressed QUBO instances indeed yield the minimal energy of the original problems, confirming that the DR reduction method preserves the optimum correctly. In order to quantify the improvement more accurately, we evaluate the samples obtained from the compressed QUBO instances using the original (uncompressed) QUBO's energy function. Inspecting the number of samples with an energy equal to or lower than the lowest sampled energies obtained from the original QUBO instances reveals that the prevalence of low-energy solutions is 17.76 (237) times higher when using the compressed BINCLUSTERING (SUBSETSUM) QUBO instances. This leads to the conclusion that DR reduction using the G_0 heuristic helps a quantum annealer to find the optimum more reliably, most likely by reducing the relative magnitude of ICE.

6.7. Concluding Remarks

QC is limited by the available hardware, and QA is no exception. While in theory, AQC is a formidable problem solving tool, its practical implementation relies largely on hybrid algorithms and heuristics. One particular limiting factor is the precision with which QUBO (or Ising) weights can be represented on QA devices. We have seen that a large value range and, at the same time, fine weight gradations leads to unwanted rounding and loss of precision.

In this chapter we discussed the role of DR in this conect, identifying it as a central factor influencing the performance of near-term QA devices. Utilizing the theoretical notion of optimum inclusion introduced in Def. 2.7, we defined the difficult task of reducing a particular problem instance's DR while maintaining at least one of its minimizing vectors. Firstly, we considered rounding as a fool-proof method of reducing the DR by forcing the smallest parameter-difference to 1. Based on this, we derived a theoretical bound on the minimal scaling factor α we need to apply before rounding in order for $\lfloor \alpha \boldsymbol{Q} \rceil \sqsubseteq \boldsymbol{Q}$. As it is not possible to efficiently compute this factor in practice, neither to directly evaluate $\boldsymbol{Q} \sqsubseteq \boldsymbol{Q}'$ as shown in Proposition 2.1, we devised a method to iteratively modify the parameters of $\boldsymbol{Q}$ while maintaining the relation. To do this, we require upper and lower bounds on the optimal energy of QUBO instances, which we use to define intervals within which a single parameter value can be varied without consequences.

Based on these findings, we devise three heuristic strategies to modify weights iteratively with the aim of reducing the overall DR in the process. In doing so, we were able to incorporate knowledge about the architecture of contemporary QA devices, which have a restricted topology and benefit from sparse QUBO instances, which the strategy G_0 exploits by preferring to set weights to 0 when possible.

To support our theoretical findings empirically, we conducted a range of experiments, showing that the DR of both randomly sampled QUBO instances and those arising from real-world problems can be reduced significantly. This effect is particularly pronounced when weight matrices are sparse. Most remarkably, we have seen that the prevalence of the globally minimizing vector in the resulting sample set of a D-Wave QA device was increased by orders of magnitude after applying DR reduction.

There are still numerous intriguing open questions and research directions waiting to be explored. The theoretical notion of optimum inclusion can be extended to an equivalence relation of QUBO instances with *the same* set of minimizing solutions, which in turn induces a partitioning on $\mathcal{Q}_n$. The number of equivalence classes is necessarily finite, bounded from above by the possible number of non-empty subsets of $\mathcal{Q}_n$, $2^{2^n}-1$, which implies there is only a finite number of meaningfully different QUBO instances of a fixed size in terms of their set of minimizing bit vectors. This further implies that, for each equivalence class, there is a representative of lowest DR. Finding these representative instances, or devising an algorithm to do so for arbitrary n, is intriguing.

The optimum energy gap we used for defining our bound α^* in Theorem 6.1 stems from the notion of spectral gaps in physics. This quantity is known to be an important factor for the efficiency of QA, leading to potentially exponential annealing times under certain conditions [71]. It is currently unknown whether there is a direct connection between optimum energy gap and DR, although our theoretical findings seem to suggest so, as a large gaps allows for stronger rounding. Besides the spectral gap, other factors influence the solution quality of QA, such as the annealing time, which we fixed to $20\mu s$ for our evaluation. An optimization toward the native connectivity structure of the particular QA device is conceivable as well. Establishing a more thorough connection between such variables, DR, and the solution quality of QA is left for future work.

Lastly, we observed that DR reduction benefits from sparsity of weight matrices, and that compressing large and simultaneously dense matrices is challenging for the iterative methods presented here. Tighter bounds may improve the performance, but they get increasingly costly to compute. A more specialized implementation using custom data structures could surely reduce the computational complexity, increasing the maximally feasible n. An even better approach would be to compute the update matrix $\boldsymbol{A}$ directly, eliminating the iterative approach entirely. Developing such an approach is left for future work, as well.

Until then, this chapter has presented a near-term method to make optimization on quantum devices more feasible and reliable by increasing the probability to find the global optimum in practice, and it has established DR as an important quantity in the context of quantum optimization.

Part IV.

Applications of Quantum-Classical Algorithms

7. Optimal Light-Source Placement

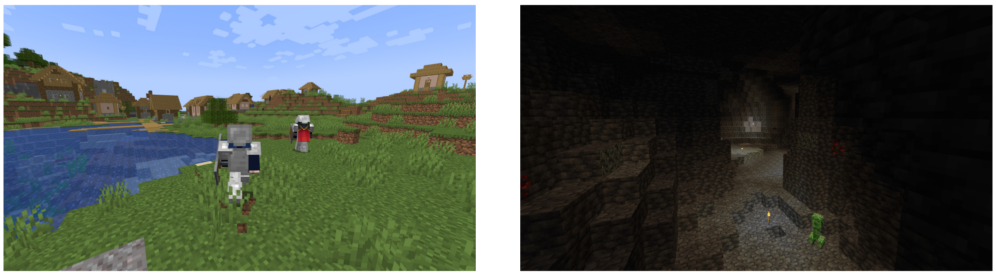
In Part II we have seen how QC can enhance classical ML algorithms, and in Part III we have reversed the direction and used classical algorithms and heuristics to improve the performance of NISQ-era QC. We have demonstrated that QA is a powerful optimization tool that can yield competitive results in the FS domain, even using today's imperfect quantum hardware. Further we have seen that quantum computers can replicate SVMs, which constitutes a step in the direction of quantum-readiness of theoretically well-founded ML models. On future quantum computers, these methods can be used *out of the box* for new (and probably bigger and more complex) data sets.

However in the real world, out-of-the-box solutions that work well and require no manual adaptation are rare to come by, which is particularly true for QC in the NISQ era. So far, there is no quantum programming language that we can use to code an algorithm, and which automatically exploits the benefits of quantum parallelism. Therefore, for the time being, we are bound to tinker at the bare circuit level, or devise QUBO formulations for use on quantum annealers on a case-by-case basis. In this part of the thesis we take a look at three real-world problems with optimization problems at their heart, which we approach with the toolset that contemporary quantum computing offers, highlighting interesting and novel techniques along the way.

This chapter is based on publication [4]. The author of this thesis developed the idea of the TORCHPLACEMENT problem, formalized it, devised the basic QUBO formulation, established the connection to SETCOVER, and wrote the majority of the paper. He also implemented the random generation and graphical representation of height maps. The experiments were designed and conducted jointly.

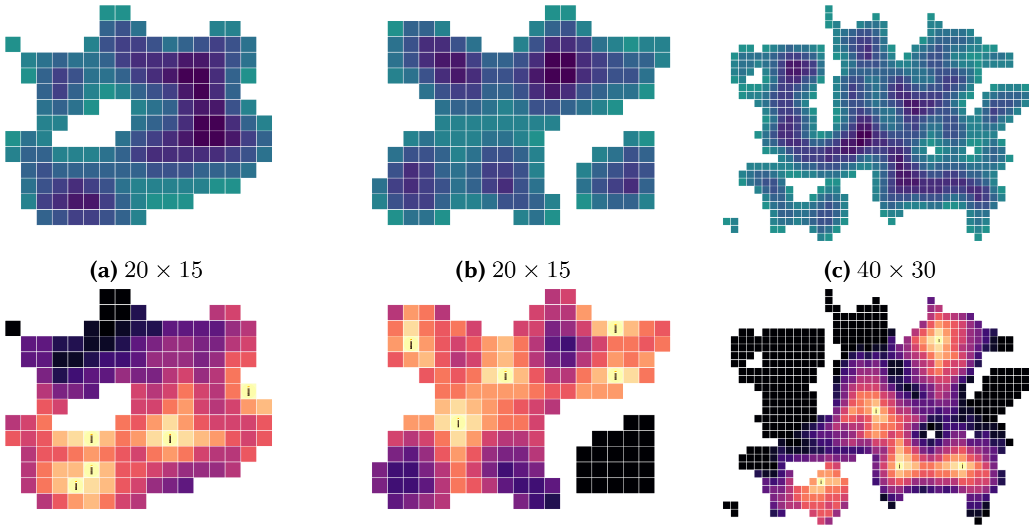
Video games are made for leisure, but at their core often contain hard optimization problems, such as maze solving or logical puzzles. Some games have even been shown to be NP-complete, such as Minesweeper [163], or PSPACE-complete, such as Sokoban [164]. A somewhat more surprising game in this context is Minecraft, developed and first published by the Swedish company Mojang around 2010[22]. It is an open-world game where players explore randomly generated worlds consisting of cubic blocks, giving the game its unique appearance (see Fig. 7.1). Blocks can be mined, collected and replaced, allowing players to build houses and machines out of hundreds of materials. Minecraft has been

[22]`https://www.minecraft.net/` (last accessed June 3, 2025)



(a) Players approaching a village. **(b)** Dimly lit cave with hostile creature.

Figure 7.1.: Screenshots of Minecraft gameplay. Source: [4].



(a) 20×15 **(b)** 20×15 **(c)** 40×30

(d) same as above, with torches **(e)** same as above, with torches **(f)** same as above, with torches

Figure 7.2.: Randomly generated heightmaps used as input data for experiments. In the top row, lighter color indicates higher elevation. In the bottom row, lighter color indicates light level. Torch symboles represent torches placed on the respective blocks. White tiles are walls. Source: [4].

one of the most successful games of the past decade and has been praised for fostering creativity in both children and adults, even in an educational context [165].

A key component of Minecraft is the exploration of dark caves. If the light level on the floor is below a certain threshold, hostile monsters spawn and attack the player. For this reason, the player places light sources like torches on the floor, which light up a surrounding area to prevent monsters from appearing. However, torches require wood and coal to make, which is why they should be used sparingly to save resources. These two competing objectives give rise to an optimization problem: How can torches be placed in a given cave environment so that all blocks are lit up sufficiently, and as few torches as possible are used?

Clearly, despite we are approaching this problem from a video game perspective, it generalizes to similar real-world setups where we want to cover a pre-defined area using as few resources as possible: Examples could be sensors, cameras, or indeed light sources with a fixed finite set of possible locations, each having a certain range across all possible locations. In every situation where *(i)* there is a fixed set of discrete locations, *(ii)* we can define a distance measure between these locations, *(iii)* a minimum distance to the nearest resource is required, and *(iv)* the number of resources should be minimized, the method described in this chapter is applicable.

Given the discrete structure of the game world of Minecraft, the first key insight is that we can use binary variables $z_s \in \mathbb{B}$ to indicate the placement of a torch at location s. As we have already seen in Chapter 3, problems of this kind can be approached naturally with QA. In this chapter, we derive a Qubo formulation for the TorchPlacement problem described above. To handle the large number of constraints, it requires Lagrangian weights, which are learned in an iterative procedure on a quantum computer, similar to [166]. In a range of experiments, we use both generated and actual game data from a Minecraft world to demonstrate the effectiveness of this method.

This chapter is structured as follows: Section 7.1 defines TorchPlacement problem formally. In Section 7.2, a Qubo formulation that attempts to solve it, combining quantum-enhanced optimization with an iterative learning scheme. In Section 7.3, we use our method to solve a range of example instances of TorchPlacement and discuss our observations. Finally, in Section 7.4 we summarize our findings.

7.1. The TorchPlacement Problem

To define the TorchPlacement formally, a mathematical description of the floor of a Minecraft cave is required. To this end, we view a case as a 3-dimensional room from a top-down perspective, consisting of cubic *blocks* that can be either empty or non-empty, resulting in a tile map where each floor tile has a certain height (see Fig. 7.2). Walls are assumed to have infinite height, and are impenetrable by light. Let $\mathcal{S} \subseteq \mathbb{Z}^3$ be the set of coordinates of floor tiles, i.e., $(i, j, k) \in \mathcal{S}$ means that at position (i, j) there is a floor tile at height k. More precisely, the elements of $\mathcal{S}$ are the coordinates of the empty blocks *above* the floor on which we could place a torch. Walls are not included in this set. Let us additionally define $\hat{\mathcal{S}} = \bigcup_{(i,j,k)\in\mathcal{S}} \bigcup_{k'=k}^{\infty} \{(i, j, k')\}$ as the whole set of empty blocks. We use the simplifying assumption that the ceiling has infinite height.

Next we need to formalize the mechanics of light spreading in Minecraft, which is based on the distance between a light source and the block in question. Each empty block in Minecraft has a discrete *light level* between 0 and 15. A torch has a light level of $L_{\text{torch}} = 14$. The distance between two blocks is given by the discrete L^1 metric, i.e., the number of blocks one has to traverse in each coordinate direction,

$$d_1((i_1, j_1, k_1), (i_2, j_2, k_2)) = |i_1 - i_2| + |j_1 - j_2| + |k_1 - k_2|.$$

When surrounded by empty space, light spreads according to this distance, diminishing by 1 level per block. That is, a block at a distance of m blocks from a light source has light level $L_{\text{torch}} - d$ (but at least 0).

However, light cannot pass through blocks. Therefore, the true light level of a block is the minimal distance between the light source moving only through empty space. We can formalize this notion recursively by defining

$$d(\boldsymbol{s}, \boldsymbol{t}) = \begin{cases} 0 & \text{if } \boldsymbol{s} = \boldsymbol{t}, \\ 1 + \min_{\boldsymbol{s}' \in \mathcal{N}(\boldsymbol{s})} d(\boldsymbol{s}', \boldsymbol{t}) & \text{otherwise,} \end{cases} \tag{7.1}$$

where $\mathcal{N}(\boldsymbol{s})$ is the set of coordinates of empty blocks around $\boldsymbol{s}$, i.e.,

$$\mathcal{N}(\boldsymbol{s}) = \{\boldsymbol{s}' \in \hat{\mathcal{S}} : \ d_1(\boldsymbol{s}', \boldsymbol{s}) = 1\}. \tag{7.2}$$

If there is no path of empty space between the two blocks, we set $d(s, t) = \infty$.

Given a heightmap as defined by a set $\mathcal{S}$, our objective is to place torches on some of those tiles in such a way that all tiles have a minimal light level of L_{min}, using as few torches as possible. We denote the set of torches as $\mathcal{T} \subseteq \mathcal{S}$ with the interpretation that if $\boldsymbol{s} \in \mathcal{T}$, then there is a torch on $\boldsymbol{s}$. Blocks containing a torch have a light level of L_{torch}. If a tile is illuminated by multiple nearby torches, it assumes the maximum over all light levels. This lets us define the light level of any empty block as

$$l(\boldsymbol{s} \mid \mathcal{T}, L_{\text{torch}}) = \max\{0, \max_{\boldsymbol{t} \in \mathcal{T}} L_{\text{torch}} - d(\boldsymbol{s}, \boldsymbol{t})\}.$$

If $\mathcal{T} = \emptyset$, then all blocks have the default light level of 0. It should be noted that this way of light spreading does not at all align with physical reality, but is a game mechanic. Where in reality we would expect light to move in straight lines from the source and be blocked or reflected by obstacles, light in Minecraft has no sense of direction, but spreads evenly to all neighboring blocks and can even move around corners this way.

The overall optimization problem we are trying to solve can now be formalized as

$$\min_{\mathcal{T} \subseteq \mathcal{S}} \quad |\mathcal{T}| \tag{7.3}$$

$$\text{s.t. } l(\boldsymbol{s} \mid \mathcal{T}, L_{\text{torch}}) \geq L_{\text{min}} \ \forall s \in \mathcal{S}. \tag{7.4}$$

For the remainder of this chapter, we set $L_{\text{torch}} = 14$ and $L_{\text{min}} = 8$, which are the same values as in Minecraft.

7.2. A Qubo formulation of TorchPlacement

TorchPlacement lends itself to Qubo, as its candidate solution space $\mathfrak{P}(\mathcal{S})$ is isomorphic to the set of binary vectors $\mathbb{B}^n$ (cf. Def. 2.3). Firstly, we need to assign an arbitrary but fixed order S to the elements of $\mathcal{S}$, such that S_i denotes the location of the i-th tile for all $i \in \{1, \dots, n\}$ with $n = |\mathcal{S}|$. A subset $\mathcal{T} \subseteq \mathcal{S}$ then corresponds to a binary vector $\boldsymbol{z} \in \mathbb{B}^n$ via $\mathcal{T} \equiv (\mathbb{1}\{S_i \in \mathcal{T}\})^{\top}_{i\in\{1,\dots,n\}}$. This way, Eq. (7.3) is simply the problem of minimizing the norm $\|\boldsymbol{z}\|_1$. It is easy to see that if we set $Q_{ii} = P \; \forall i \in \{1, \dots, n\}$ with some arbitrary positive penalty value $P > 0$, then $f_{\boldsymbol{Q}}(\boldsymbol{z}) = P\|\boldsymbol{z}\|_1$, which is minimized by $\mathbf{0}$.

The constraints defined by Eq. (7.4) are much more challenging to embed into the QUBO instance. Given a candidate solution $\boldsymbol{z}$, we need to ensure that the light level is at least L_{min} at every location in $\mathcal{S}$. To ensure this, the torch's distance can be at most $L_{\text{torch}} - L_{\text{min}}$,

$$\min_{\substack{j\in\{1,\dots,n\},\\ z_j=1}} d(S_i, S_j) \leq L_{\text{torch}} - L_{\text{min}} \; \forall i \in \{1, \dots, n\}. \tag{7.5}$$

QUBO can natively only encode quadratic functions, but max is a non-linear operation, which makes these types of constraints a challenge.

Avoiding Non-Linearity In theory, there are several smooth approximations of the min and max functions which we could use here. However, we can simplify the constraints considerably by realizing that they constitute, in essence, a binary condition on every block, namely whether it is lit up sufficiently or not. To encode this, we define the binary matrix $\boldsymbol{D} \in \mathbb{B}^{n\times n}$ with entries D_{ij} given by

$$D_{ij} = \mathbb{1}\{d(S_i, S_j) \leq L_{\text{torch}} - L_{\text{min}}\} \; \forall i, j \in \{1, \dots, n\}.$$

To ensure that $\min_j d(S_i, S_j) \leq L_{\text{torch}} - L_{\text{min}}$, we can check whether

$$\sum_j x_j D_{ij} = (\boldsymbol{D}\boldsymbol{z})_i \geq 1 \;, \tag{7.6}$$

which immediately yields linear constraints and circumvents non-linear constraints entirely.

The new formulation of the optimization problem in Eqs. (7.3) and (7.4) enhanced with Eq. (7.6) reads

$$\min_{\boldsymbol{z}\in\mathbb{B}^n} \mathbf{1}^{\top}\boldsymbol{z} \tag{7.7}$$

$$\text{s.t. } \boldsymbol{D}\boldsymbol{z} \mathrel{\dot{\geq}} \mathbf{1}_n, \tag{7.8}$$

where $\dot{\geq}$ denotes element-wise $\geq$, yielding n separate inequality constraints.

Similarity with SETCOVER By realizing that every block $\boldsymbol{s} \in \mathcal{S}$ defines itself a subset of $\mathcal{S}$ containing all blocks that would be lit up sufficiently by a torch on $\boldsymbol{s}$, we find that TORCHPLACEMENT has striking similarity to the NP-hard SETCOVER problem (see, e.g.,

[167]): In SetCover, we are given a set A and a collection of subsets B_i with $B_i \subseteq A$ for each $i \in I$, and $\bigcup_{i \in I} B_i = A$. The objective is to find a $J \subseteq I$ such that $|J|$ is minimal and $\bigcup_{j \in J} B_j = A$. Given a heightmap with tiles $\mathcal{S}$ and distance function d as defined in Eq. (7.1), we set $A = \mathcal{S}$, $I = \{1, \dots, n\}$ and $B_i = \{s \in \mathcal{S} : \ d(S_i, s) \leq L_{\text{torch}} - L_{\text{min}}\}$ for all $i \in \{1, \dots, n\}$. Thus, a solution $J \subseteq I$ with minimal $|J|$ is a minimal set of torches that illuminates all other tiles.

7.2.1. Handling Inequality Constraints

It remains to show how to obtain a valid Qubo instance from the constrained problem given in Eqs. (7.7) and (7.8). The linear inequality constraint in Eq. (7.8) can be reformulated using an auxiliary vector $\boldsymbol{m} \in \mathbb{N}_0^n$ with non-negative integer entries through $\boldsymbol{Dz} \dot{\geq} \boldsymbol{1} \ \Leftrightarrow \boldsymbol{Dz} - \boldsymbol{1} = \boldsymbol{m}$, leading to an equivalent problem formulation,

$$\min_{\boldsymbol{z} \in \{0,1\}^n} \ \boldsymbol{1}^\intercal \boldsymbol{z} \text{ s.t. } \boldsymbol{Dz} - \boldsymbol{1} - \boldsymbol{m} = 0, \ \boldsymbol{m} \in \mathbb{N}_0^n. \tag{7.9}$$

We can approach such a problem containing equality constraints by introducing a matrix of binary slack variables $\boldsymbol{S} \in \mathbb{B}^{n \times m}$, where $m = \lceil \log_2 n \rceil$ [168]. If we define a vector $\boldsymbol{r} = (2^0, 2^1, \dots, 2^{m-1})^\intercal$, then we find that $\boldsymbol{m} = \boldsymbol{Sr}$. The constrained problem in Eq. (7.9) can then be turned into an equivalent unconstrained problem by introducing a penalty value $\beta > 0$ and solving

$$\min_{\boldsymbol{z} \in \{0,1\}^n, \, \boldsymbol{S} \in \{0,1\}^{n \times m}} \ \boldsymbol{1}^\intercal \boldsymbol{z} + \beta \left\| \boldsymbol{Dz} - \boldsymbol{1} - \boldsymbol{Sr} \right\|^2 . \tag{7.10}$$

This formulation has the disadvantage of introducing nm additional binary variables, increasing the Qubo dimension to $n(m+1)$, which quickly exceeds the capabilities of today's QA devices. We therefore use a different Qubo formulation that uses fewer qubits by using the iterative method Alternating Direction Method of Multipliers (ADMM) [169]. Firstly, we establish a new problem formulation

$$\begin{aligned} \min_{\boldsymbol{z} \in \mathbb{B}^n, \, \boldsymbol{m} \in \mathbb{Z}^n} \ & \boldsymbol{1}^\intercal \boldsymbol{z} + \gamma \boldsymbol{1}^\intercal \Theta(\boldsymbol{m}) \\ \text{s.t. } & \boldsymbol{c}\left(\boldsymbol{z}, \boldsymbol{m}\right) = 0, \end{aligned} \tag{7.11}$$

with $\gamma > 0$, $\boldsymbol{c}\left(\boldsymbol{z}, \boldsymbol{m}\right) = \boldsymbol{Dz} - \boldsymbol{1} - \boldsymbol{m}$, and Θ being an element-wise step function defined as $\Theta(\boldsymbol{m}) = (\mathbb{1}\{m_1 < 0\}, \dots, \mathbb{1}\{m_n < 0\})$. Here, $\boldsymbol{1}^\intercal \Theta(\boldsymbol{m})$ is a penalty term punishing negative entries of $\boldsymbol{m}$, since we want to ensure $\boldsymbol{m} \in \mathbb{N}_0^n$. Now, vectors $\boldsymbol{z} \in \mathbb{B}^n$ and $\boldsymbol{m} \in \mathbb{Z}^n$ satisfying Eq. (7.11) are also optimal for the problem in Eq. (7.9). Next we introduce the augmented Lagrangian [170, 171]

$$L\left(\boldsymbol{z}, \boldsymbol{m}, \boldsymbol{\lambda}, \mu\right) = \boldsymbol{1}^\intercal \boldsymbol{z} + \gamma \boldsymbol{1}^\intercal \Theta(\boldsymbol{m}) + \boldsymbol{\lambda}^\intercal \boldsymbol{c}\left(\boldsymbol{z}, \boldsymbol{m}\right) + \frac{\mu}{2} \left\| \boldsymbol{c}\left(\boldsymbol{z}, \boldsymbol{m}\right) \right\|^2 , \tag{7.12}$$

where $\boldsymbol{\lambda}$ and μ are coefficients and multipliers for the penalty terms. To minimize this expression, we use ADMM as outlined in Algorithm 7.1, which updates the vectors $\boldsymbol{z}$ and $\boldsymbol{m}$ iteratively. Expanding line 5 yields

Algorithm 7.1 Alternating Direction Method of Multipliers (ADMM). Source: [4].

```
Input:   Initial $\mu_0 > 0$,
Output:  $z^*$, $m^*$, $\lambda^*$, $\mu^*$ optimizing $L(z, m, \lambda, \mu)$
1: $m^* \leftarrow 0$
2: $\lambda^* \leftarrow 0$
3: $\mu^* \leftarrow \mu_0$
4: repeat
5:     $z^* \leftarrow \arg\min_z L(z, m^*, \lambda^*, \mu^*)$ {QA applicable}
6:     $m^* \leftarrow \arg\min_m L(z^*, m, \lambda^*, \mu^*)$
7:     $\lambda^* \leftarrow \lambda^* + \mu^* c(z, m)$
8:     Update $\mu^*$
9: until a convergence criterium is met
```

$$\begin{aligned}\underset{\boldsymbol{z}\in\mathbb{B}^n}{\arg\min}\ L(\boldsymbol{z},\boldsymbol{m},\boldsymbol{\lambda},\mu) &= \underset{\boldsymbol{z}\in\{0,1\}^n}{\arg\min}\ \mathbf{1}^\intercal\boldsymbol{z} + \boldsymbol{\lambda}^\intercal\boldsymbol{c}(\boldsymbol{z},\boldsymbol{m}) + \frac{\mu}{2}\left\|\boldsymbol{c}(\boldsymbol{z},\boldsymbol{m})\right\|^2 \\ &= \underset{\boldsymbol{z}\in\{0,1\}^n}{\arg\min}\ \mathbf{1}^\intercal\boldsymbol{z} + \boldsymbol{\lambda}^\intercal\boldsymbol{D}\boldsymbol{z} + \frac{\mu}{2}\left(\boldsymbol{z}^\intercal\boldsymbol{D}^\intercal\boldsymbol{D}\boldsymbol{z} - (\mathbf{1}+\boldsymbol{m})^\intercal\boldsymbol{D}\boldsymbol{z}\right),\end{aligned}$$

corresponding to a Qubo formulation solvable using QA. Finally, line 6 can be reduced to

$$\begin{aligned}\underset{\boldsymbol{m}\in\mathbb{Z}^n}{\arg\min}\ L(\boldsymbol{z},\boldsymbol{m},\boldsymbol{\lambda},\mu) &= \underset{\boldsymbol{m}\in\mathbb{Z}^n}{\arg\min}\ \gamma\mathbf{1}^\intercal\Theta(\boldsymbol{m}) + \boldsymbol{\lambda}^\intercal\boldsymbol{c}(\boldsymbol{z},\boldsymbol{m}) + \frac{\mu}{2}\left\|\boldsymbol{c}(\boldsymbol{z},\boldsymbol{m})\right\|^2 \\ &= \underset{\boldsymbol{m}\in\mathbb{Z}^n}{\arg\min}\ \gamma\mathbf{1}^\intercal\Theta(\boldsymbol{m}) + \frac{\mu}{2}\left\|\boldsymbol{D}\boldsymbol{z} - \mathbf{1} - \boldsymbol{m} + \frac{1}{\mu}\boldsymbol{\lambda}\right\|^2 \\ &= \max\{\mathbf{0},\ \boldsymbol{D}\boldsymbol{z} - \mathbf{1}\},\end{aligned}$$

where max is taken element-wise in the last line. As an update rule for μ_k^* in line 8 we choose

$$\mu_{t+1} = \begin{cases}\rho\mu_t & \text{if } \left\|\boldsymbol{c}(\boldsymbol{z}_t,\boldsymbol{m}_t)\right\| > 10\mu_t\left\|\boldsymbol{D}(\boldsymbol{m}_t - \boldsymbol{m}_{t+1})\right\|, \\ \mu_t/\rho & \text{if } \left\|\boldsymbol{D}(\boldsymbol{m}_t - \boldsymbol{m}_{t+1})\right\| > 10\mu_t\left\|\boldsymbol{c}(\boldsymbol{z}_t,\boldsymbol{m}_t)\right\|, \\ \mu_t & \text{otherwise,}\end{cases}$$

using a fixed learning rate $\rho \geq 1$, following [169]. The subscript t denotes the state of $\boldsymbol{z}^*$, $\boldsymbol{m}^*$, $\boldsymbol{\lambda}^*$ and μ^* after iteration t of ADMM. In place of a convergence criterium in line 9 we use a fixed budget of N of calls to the quantum computer.

To summarize, in order to solve Eqs. (7.7) and (7.8) we now solve a sequence of Qubo problems, updating the parameters $\boldsymbol{m}$, $\boldsymbol{\lambda}$ and μ according to Algorithm 7.1. As we use QA for solving the Qubo instances, the resulting algorithm can be described as a hybrid quantum-classical approach.

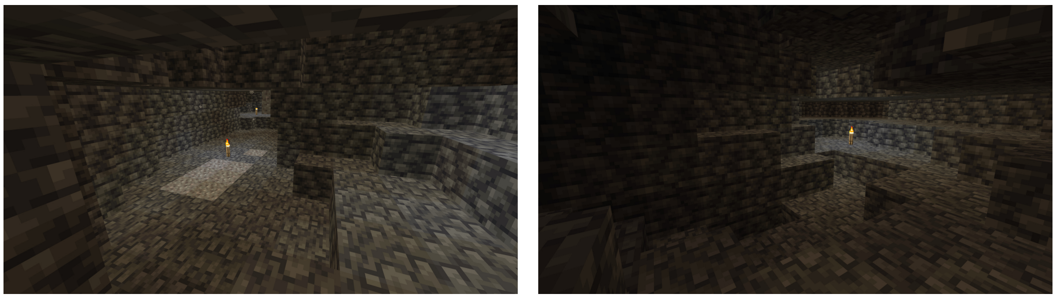


Figure 7.3.: Screenshots of Minecraft caves. In Section 7.3, heightmaps based on these caves are used for the experimental evaluation. Source: [4].

7.3. Experimental Evaluation

We evaluate the ADMM-based light placement procedure on a range heightmaps. Most were generated randomly using 2-dimensional Perlin noise [172], ranging in size between 149 and 768 floor tiles, at an average of about 430 tiles. Others were extracted from cave sections found in an actual Minecraft world. For these experiments, in order to obtain a more comprehensive overview of the performance of different QUBO solvers, we compare four different solvers provided by the D-Wave Ocean[23] Python package: Simulated Annealing (SA), Tabu search (Tabu), a combination of those two (TabuSA), and a real QA, namely a D-Wave Advantage System 5.4 with 5614 qubits and a total of $40,050$ couplers. The combination of SA and Tabu is achieved by using the D-Wave Hybrid[24] Python package, which provides a parallelized implementation. We use the default parameters provided by D-Wave for all solvers. As a pre-processing step, we apply weight compression using the method presented in Chapter 6, which leads to improved solution quality for QA. We repeat ADMM 10 times for every combination of solver and heightmap, plotting mean performance and 95%-confidence intervals. The hyperparameter settings $\mu_0 = 0.01$ and $\rho = 1.1$ were found to yield the best results.

Real Minecraft Caves Figure 7.3 (right) shows an in-game view of a real Minecraft cave section, which we extracted a heightmap with $n = 67$ floor blocks from. Figure 7.4 shows the performance of all QUBO solvers for this cave: We plot the number of torches as well as the number of violated constraints over the iterations of ADMM. Evidently, all QUBO solvers quickly converge to solutions violating no constraints. QA places the most torches, indicating that its solution quality falls behind the other methods, which is a common problem of NISQ devices discussed thoroughly in previous chapters. Nonetheless, Fig. 7.5 shows the result of a particular run using QA after 2, 5 and 10 iterations of ADMM. After 10 iterations, an optimal solution is found, such that every block is lit.

[23]`https://docs.ocean.dwavesys.com/en/stable` (last accessed June 3, 2025)
[24]`https://docs.ocean.dwavesys.com/projects/hybrid/en/stable` (last accessed June 3, 2025)

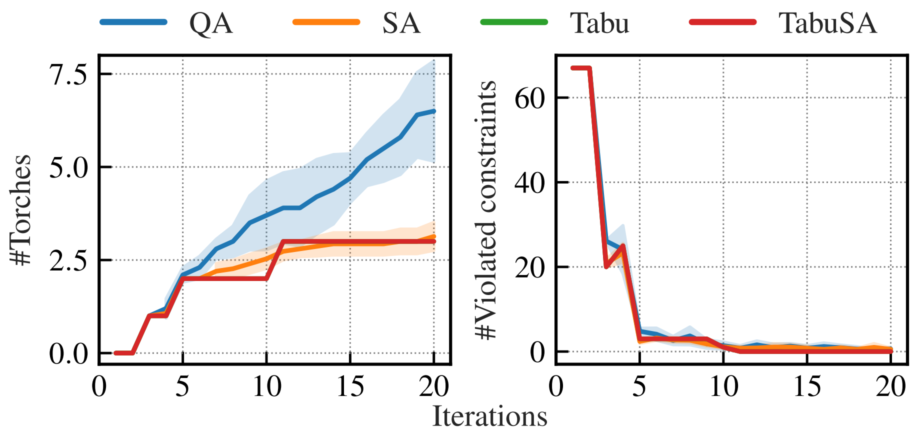


Figure 7.4.: Performance of the four QUBO solvers QA, SA, Tabu and TabuSA on the real Minecraft cave depicted in Fig. 7.3 (bottom). Source: [4].

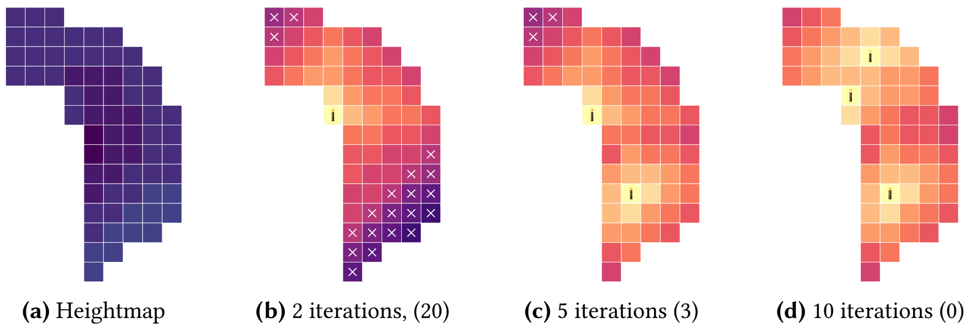


Figure 7.5.: Real Minecraft cave (Fig. 7.3, right) with $n = 67$; white crosses signify light level below L_{min}. Number of constraint violations in parentheses. The QUBO instances were solved using QA. Source: [4].

For problems with large n, we use TabuSA from this point onward, since the results obtained with QA were not satisfactory. In Fig. 7.6 we see solutions after a certain number of ADMM iterations on the heightmaps of the other real Minecraft cave shown in Fig. 7.3 (left), which has $n = 195$ floor block. Again, we observe that with more iterations lead to more and more torches being place until the constraints are fulfilled. In Figs. 7.7 to 7.10 we depict ADMM solutions for random heightmaps of varying sizes, again using the TabuSA solver. Each plot shows the original height map, two intermediate results during the Lagrangian learning phase, and the final result after the iteration budget is depleted. The resulting torch placements are all valid, violating no constraints, and the torches seem to be placed as sparsely as possible.

We find that some solutions contain two torches being placed right next to each other. While this seems inefficient, the objective of our optimization problem does not explicitly punish torches being place close to each other. On the contrary, certain heightmap layouts

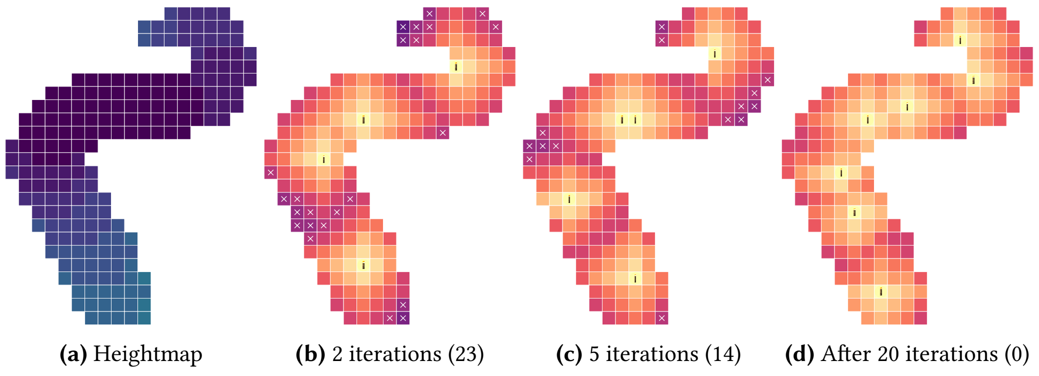

(a) Heightmap **(b)** 2 iterations (23) **(c)** 5 iterations (14) **(d)** After 20 iterations (0)

Figure 7.6.: Real Minecraft cave (Fig. 7.3, left) with $n = 195$; white crosses signify light level below L_{min}. Number of constraint violations in parentheses. The QUBO instances were solved with TabuSA. Source: [4].

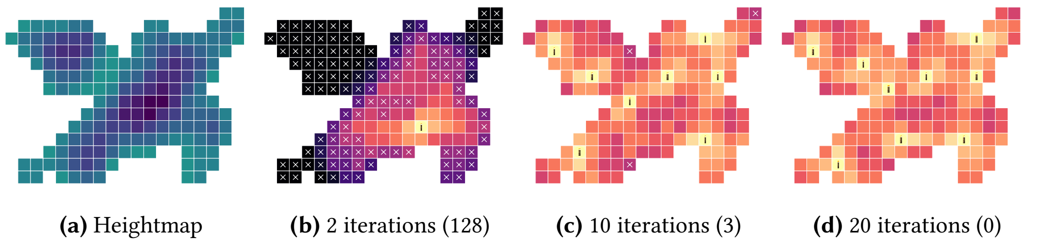

(a) Heightmap **(b)** 2 iterations (128) **(c)** 10 iterations (3) **(d)** 20 iterations (0)

Figure 7.7.: Generated Minecraft cave with $n = 168$; white crosses signify light level below L_{min}. Number of constraint violations in parentheses. Source: [4].

do in fact allow or even require adjacent torches, e.g., when they are at vastly different height levels. Also, when the light of a torch does not quite reach into a corner and another torch must be placed, its exact placement is arbitrary.

7.4. Concluding Remarks

In this chapter we discussed the TorchPlacement problem arising from the video game Minecraft, how it is related to SetCover, and how it can be solved using a Qubo formulation enhanced with an iterative procedure based on ADMM. While other methods for solving SetCover using Qubo require a large number of auxiliary variables [173], the approach shown here uses Lagrangian multipliers that are updated iteratively, shifting the complexity from additional variables to an additional outer optimization loop, saving valuable quantum resources when using QA as a solver. The results demonstrate that the method yields satisfactory results for heightmaps with up to 700 floor blocks. However, NISQ limitations restricted the experiments on an actual quantum annealer to heightmaps of only up to $n \leq 100$ blocks.

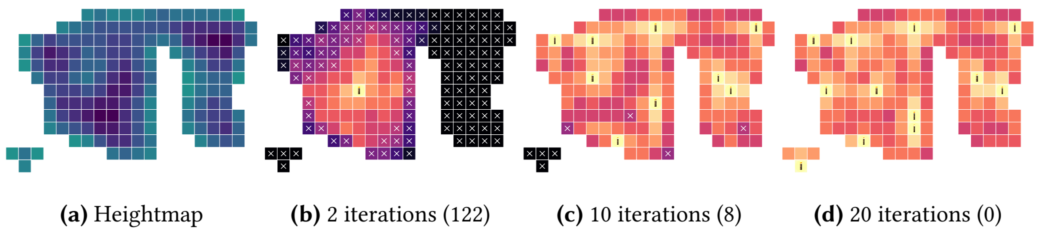

(a) Heightmap (b) 2 iterations (122) (c) 10 iterations (8) (d) 20 iterations (0)

Figure 7.8.: Generated Minecraft cave with $n = 169$; white crosses signify light level below L_{min}. Number of constraint violations in parentheses. Source: [4].

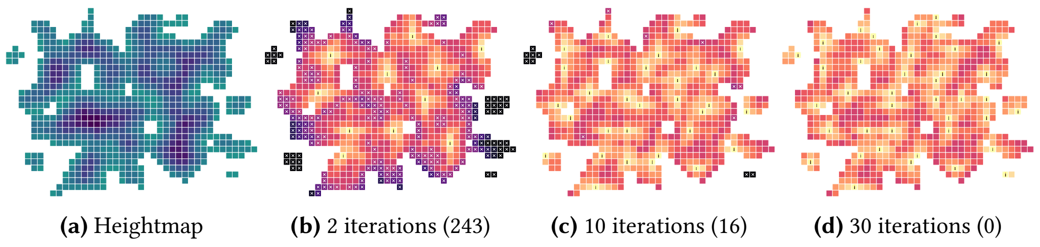

(a) Heightmap (b) 2 iterations (243) (c) 10 iterations (16) (d) 30 iterations (0)

Figure 7.9.: Generated Minecraft cave with $n = 711$; white crosses signify light level below L_{min}. Number of constraint violations in parentheses. Source: [4].

Games and their computational complexity have inspired research for decades [163, 174, 175], and Minecraft is no exception. The connection between TorchPlacement and Set-Cover is not obvious, and developing new solution strategies for one problem might light to new insights in other areas. For instance, we find that TorchPlacement and Set-Cover are similar to other problems such as MaximumCoverage, where an upper limit k is given, and (staying in the context of TorchPlacement) we are looking to light up the largest possible area using at most k torches.

The method described in this chapter may well be the first instance of QC being applied to a problem arising from Minecraft. QC solutions to problems appearing in video games have real potential to guide the application of quantum optimization to real-word problems, as they illustrative how to approach problems from a QC perspective. In addition,

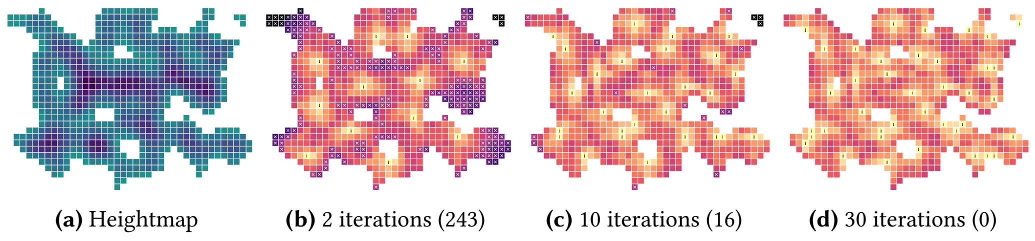

(a) Heightmap (b) 2 iterations (243) (c) 10 iterations (16) (d) 30 iterations (0)

Figure 7.10.: Generated Minecraft cave with $n = 708$; white crosses signify light level below L_{min}. Number of constraint violations in parentheses. Source: [4].

combining optimization, QC, and video games may help to make scientific topics more intuitive and approachable for young scientists.

8. Solving Sudoku using Quantum Annealing

In the previous chapter we have seen an application of QA as a component within a bigger optimization framework, where multiple QUBO instances had to be solved to obtain a solution. In many cases, however, it is possible to embed an optimization problem as a whole and obtain a globally optimal solution in a single run. As we will see in this chapter, this is even possible if the search space is highly constrained, despite the fact that QUBO is, as its name implies, unconstrained by default. In Section 2.2 we described how constraints can be incorporated by introducing penalty weights that drastically increase the energy of forbidden solutions, rendering them non-optimal. We already applied this technique in Section 4.1, where we added a penalty λ for ensuring that $\sum_i \alpha_i y_i = 0$. In Chapter 7, we took the concept to the extreme by using an entire vector $\boldsymbol{\lambda}$ of Lagrange multipliers, which we learned through ADMM in an outer optimization loop. Disadvantages of highly constrained QUBO instances are *(i)* an increased DR due to the large magnitude of penalty weights, and *(ii)* a much *spikier* energy landscape due to valid solutions being hidden in *valleys* surrounded by penalized solutions. We addressed the first point in Chapter 6 with our DR reduction methods, and we address the second point in this chapter through another real-world application, the puzzle game Sudoku.

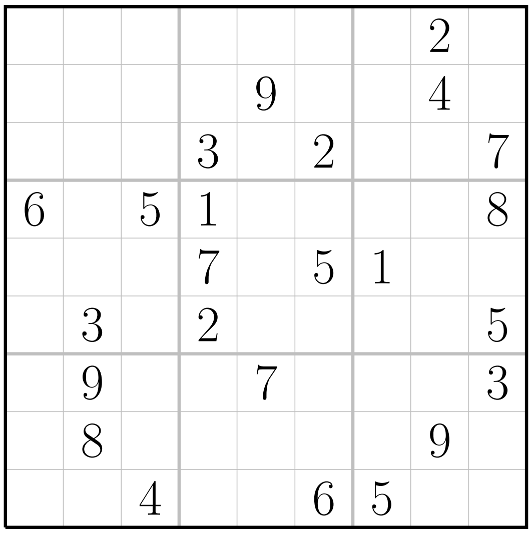
This chapter is based on the publication [3] by the author of this thesis.

The Japanese puzzle game Sudoku consists of a 9×9 square grid of cells, each of which can hold a number from 1 to 9. Some cells contain *clues*, but most are initially empty. The objective of this game is to fill in the missing numbers in a specific way, governed by a set of rules:

Definition 8.1 (Sudoku). *Given a* 9×9 *grid of cells, a valid Sudoku fulfills that*

1. *each row contains all numbers exactly once,*
2. *each column contains all numbers exactly once,*
3. *each* 3×3 *sub-grid (*block*) contains all numbers exactly once.*

The generalized version of Sudoku with grids of size $N^2 \times N^2$ has been proven to be NP-complete [176]. In this chapter we will develop a QUBO embedding for Sudoku.



							2	
				9			4	
			3		2			7
6		5	1					8
			7		5	1		
	3		2					5
	9			7				3
	8						9	
		4			6	5		

Figure 8.1.: Hard Sudoku puzzle from the New York Times on January 8, 2024, containing 24 clues. Source: [3].

The original idea of embedding Sudoku in QUBO was inspired by [177], where Hopfield Networks are used to solve the generalized variant of this game. In that work, constraints are employed to ensure that all conditions of Def. 8.1 are fulfilled, and the initial clues are used correctly. This approach yields a valid, yet highly constrained QUBO instance, meaning that bit vectors violating the constraints outnumber those that fulfill the constraints by a large margin. Considering a one-hot encoding of all numbers on the Sudoku grid (ignoring all further constraints for the moment) leads to $9^3 = 729$ bits (9 bits per cell, 9×9 cells) representing 10^{81} possible assignments (including the empty cell) versus 2^{729} possible bit vectors. This means that the proportion of valid one-hot encodings of Sudoku is $10^{81}/2^{729} \approx 10^{-138}$.

The approach in this chapter deviates from [177] in two ways: Instead of additional constraints for the initial clues, the corresponding variables are fixed and removed them from the optimization altogether, reducing the problem size considerably. Moreover, no constraints are used to force all cells to be non-empty, but non-empty cells are encouraged through rewards, relying on the QUBO solver to discover the correct solutions of lowest energy. While [177] presents a general solution for any N, this chapter focuses on $N = 3$ as the default Sudoku. However, generalizing the techniques presented here can be easily achieved using the same techniques.

The QUBO formulation described below closely aligns with [178], where a Sudoku is mapped to a graph and interpreted as an instance of the INDEPENDENT SET problem. While in [178], pre-processing steps are performed on the Sudoku puzzle itself to simplify the resulting QUBO instance, the following approach incorporates clues and simplification on the QUBO level using techniques discussed in previous chapters as well as new ones.

8.1. The Sudoku QUBO

We start by defining the one-hot-based QUBO formulation of Sudoku mentioned above with its extremely small proportion of valid bit vectors. This problem is going to be mitigated later on.

Firstly, we introduce binary indicator variables $s_{ijk} \in \mathbb{B}$ for all $i, j, k \in \{1, \dots, 9\}$ with the interpretation that $s_{ijk} = 1$ means "the cell in row i, column j contains number k", resulting in a total of 729 variables describing a Sudoku grid that can hold every possible combination of numbers. The constraints of Def. 8.1 are encoded into a QUBO weight matrix $\boldsymbol{S} \in \mathcal{Q}_{729}$ through a positive penalty weight $\lambda > 0$. Given two cells represented by (i, j, k) and (i', j', k'), a penalty is added if any of the following four conditions hold:

(i) $i = i' \ \wedge \ j \neq j' \ \wedge \ k = k'$
(same number twice in the same row)

(ii) $i \neq i' \ \wedge \ j = j' \ \wedge \ k = k'$
(same number twice in the same column)

(iii) $\lfloor i/3 \rfloor = \lfloor i'/3 \rfloor \ \wedge \ \lfloor j/3 \rfloor = \lfloor j'/3 \rfloor \ \wedge \ k = k'$
(same number twice in the same block)

(iv) $i = i' \ \wedge \ j = j' \ \wedge \ k \neq k'$
(multiple numbers in the same cell)

If we were to use penalties only, empty cells would be assigned the same energy as filled cells containing numbers that fulfill the constraints. To solve this problem, we reward solutions with a high number of 1-bits by giving each s_{ijk} a linear weight of -1. To represent all s_{ijk} as one continuous bit vector, we need an arbitrary but fixed bijection $\iota : \{1, \dots, 9\}^3 \to \{1, \dots, 729\}$, such as

$$\iota(i, j, k) = 81i + 9j + k - 90. \tag{8.1}$$

This way, we can use continuous binary vectors $\boldsymbol{z} \in \mathbb{B}^{729}$ to represent the binary variables through $z_u = s_{ijk}$ if $\iota(i, j, k) = u$ for all $u \in \{1, \dots, 729\}$. The entries of the resulting QUBO matrix $\boldsymbol{S}$ can be summarized as

$$S_{uv} = \begin{cases} \lambda & \text{if } u \neq v \text{ and any of (i)-(iv) are true, where } u = \iota(i, j, k) \text{ and } v = \iota(i', j', k'), \\ -1 & \text{if } u = v, \\ 0 & \text{otherwise.} \end{cases} \tag{8.2}$$

If we use the specific bijection ι given in Eq. (8.1), the matrix $\boldsymbol{S}$ (in its symmetrical form) can be expressed quite beautifully as

$$\begin{aligned} \boldsymbol{S} &= \lambda[(1 - \boldsymbol{I}_9)^{\oplus 3} + \boldsymbol{K}^{\otimes 2} \otimes \boldsymbol{I}_9] - \boldsymbol{I}_9^{\otimes 3} \\ \text{with } \boldsymbol{K} &= \boldsymbol{I}_3 \otimes (1 - \boldsymbol{I}_3), \end{aligned}$$

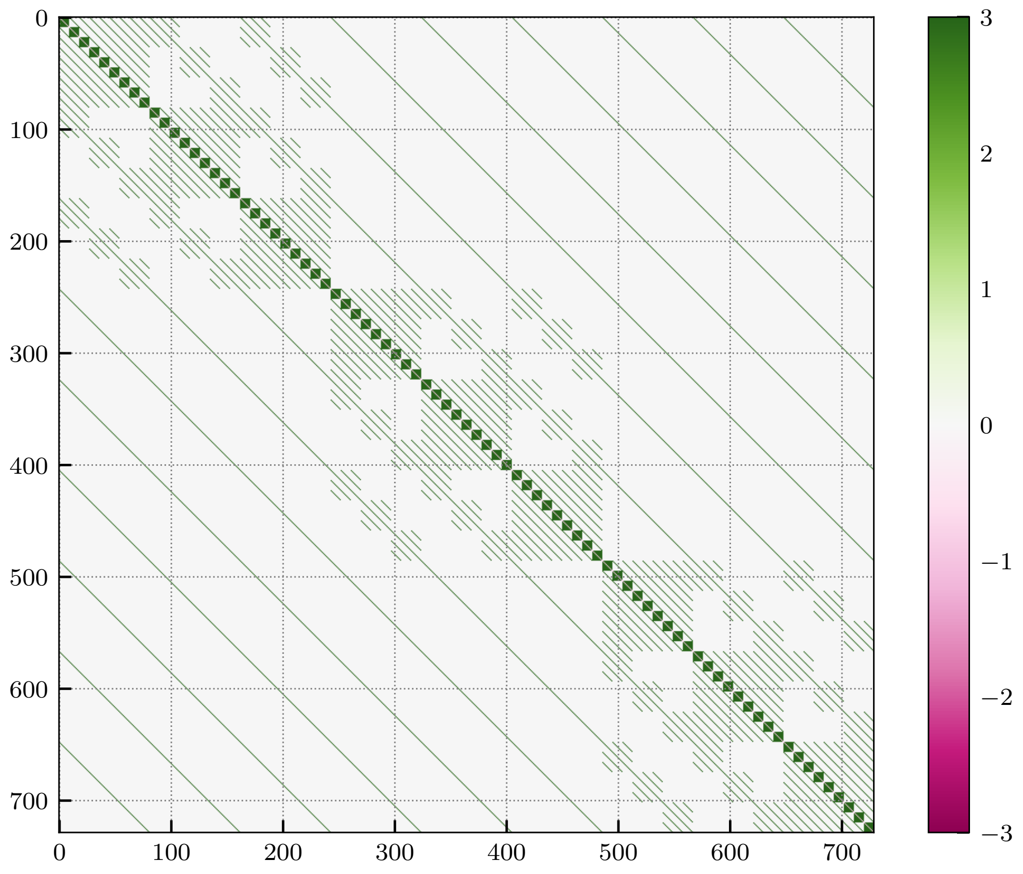


Figure 8.2.: Sudoku QUBO weight matrix $\boldsymbol{S}$ in its symmetrical form, as defined in Eq. (8.2). Source: [3].

where $\oplus$ denotes the Kronecker sum defined as $\boldsymbol{A} \oplus \boldsymbol{B} = \boldsymbol{A} \otimes \boldsymbol{I}_b + \boldsymbol{I}_a \otimes \boldsymbol{B}$ for square matrices $\boldsymbol{A} \in K^{a \times a}$ and $\boldsymbol{B} \in K^{b \times b}$ over any field K (not to be confused with the direct sum used in Def. 4.4), and $\otimes$ is the usual Kronecker product.

The value of the penalty weight λ has to be chosen such that it cancels out the reward of two incorrectly placed numbers. As each single number placement yields a reward of -1, this implies $\lambda \geq 2$. For the remainder of this chapter, we simply set $\lambda = 3$. The full matrix $\boldsymbol{S}$ is shown in Fig. 8.2.

8.2. Incorporating Clues

A Sudoku puzzle comes with a number of clues, i.e., pre-filled cells, and a human players deduces the missing numbers logically. Let $C \subseteq \{1, \ldots, 9\}^3$ denote a set of tuples (i, j, k) which are given as clues. Instead of resorting to additional clues to force the grid cells to aling with C, we can fix the values of the corresponding variables at indices $\iota(i,j,k)$ for every $(i,j,k) \in C$ and exclude them from the optimization procedure altogether. With every variable that is removed (i.e., with every clue), the search space size is reduced by half.

8.2.1. Saving Quantum Resources through Clamping

Any Qubo instance $\boldsymbol{Q}$ of size n can be transformed into a smaller instance $\boldsymbol{Q}'$ by assigning fixed values to one or more variables, which is sometimes referred to as *clamping* [102]. Let $I_0, I_1 \subseteq \{1,\dots,n\}$ with $I_0 \cap I_1 = \emptyset$. We want to implicitly assign $z_i = 0\ \forall i \in I_0$ and $z_j = 1\ \forall j \in I_1$, which allows us to eliminate these variables, such that $\boldsymbol{Q}'$ has only size $m = n - |I_0| - |I_1|$. First, we define an injection κ mapping the remaining variable indices $\{1,\dots,n\}\backslash I_0 \backslash I_1$ to $\{1,\dots,m\}$. Now, for any vector $\boldsymbol{z}' \in \mathbb{B}^m$, the original energy value can be computed by re-inserting the implicit bits into $\boldsymbol{z}'$ and evaluating with the original $\boldsymbol{Q}$. For simplicity, assume that $\boldsymbol{z}'$ is padded with a constant 1 at the end, obtaining $\boldsymbol{z}'' = (x_1', \dots, x_m', 1) \in \mathbb{B}^{m+1}$. This allows us to construct a matrix $\boldsymbol{T} \in \mathbb{B}^{n\times(m+1)}$ that re-inserts the implicit bits, whose rows read

$$\boldsymbol{T}_{i,\cdot} = \begin{cases} (0,0,\dots,0,0) & \text{if } i \in I_0, \\ (0,0,\dots,0,1) & \text{if } i \in I_1, \\ \boldsymbol{e}_{\kappa(i)}^\intercal & \text{otherwise.} \end{cases} \tag{8.3}$$

This lets us restore the original energy value as

$$f_{\boldsymbol{Q}}(\boldsymbol{z}) = \underbrace{(\boldsymbol{T}\boldsymbol{z}'')^\intercal}_{\boldsymbol{z}^\intercal} \boldsymbol{Q} \underbrace{(\boldsymbol{T}\boldsymbol{z}'')}_{\boldsymbol{z}} = \boldsymbol{z}''^\intercal \underbrace{\boldsymbol{T}^\intercal\boldsymbol{Q}\boldsymbol{T}}_{\boldsymbol{Q}''} \boldsymbol{z}''. \tag{8.4}$$

As a final step, we can exploit the fact that $\boldsymbol{T}^\intercal\boldsymbol{Q}\boldsymbol{T} = \boldsymbol{Q}'' \in \mathcal{Q}m+1$ is only applied to vectors with a constant 1 at the end, i.e., $x''_{m+1} = 1$, in order to reduce its size by 1: We set $\boldsymbol{Q}' = \boldsymbol{Q}''_{:m,:m} + \operatorname{diag}\big[\boldsymbol{Q}''_{m+1,:m}\big] + \operatorname{diag}\big[\boldsymbol{Q}''_{:m,m+1}\big] \in \mathbb{R}^{m\times m}$, using $z_i z_{m+1} = z_i\ \forall i \in \{1,\dots,m\}$. The notation $:m$ as an index denotes all rows or columns up to and including index m, as illustrated in Fig. 8.3. The remaining value $Q''_{m+1,m+1} = c$ is needed to recover the original energy value

$$f_{\boldsymbol{Q}}(\boldsymbol{z}) = f_{\boldsymbol{Q}'}(\boldsymbol{z}') + c = \boldsymbol{z}'^\intercal\boldsymbol{Q}'\boldsymbol{z}' + c, \tag{8.5}$$

but can be ignored during optimization.

With the set of clues C at hand, the relevant variables of $\boldsymbol{S}$ can now be clamped the following way: For all $(i,j,k) \in C$,

(I) clamp $z_{\iota(i,j,k)} = 1$
(cell (i,j) contains the correct value)

(II) $\forall k' \neq k$, clamp $z_{\iota(i,j,k')} = 0$
(remove all other values for cell (i,j))

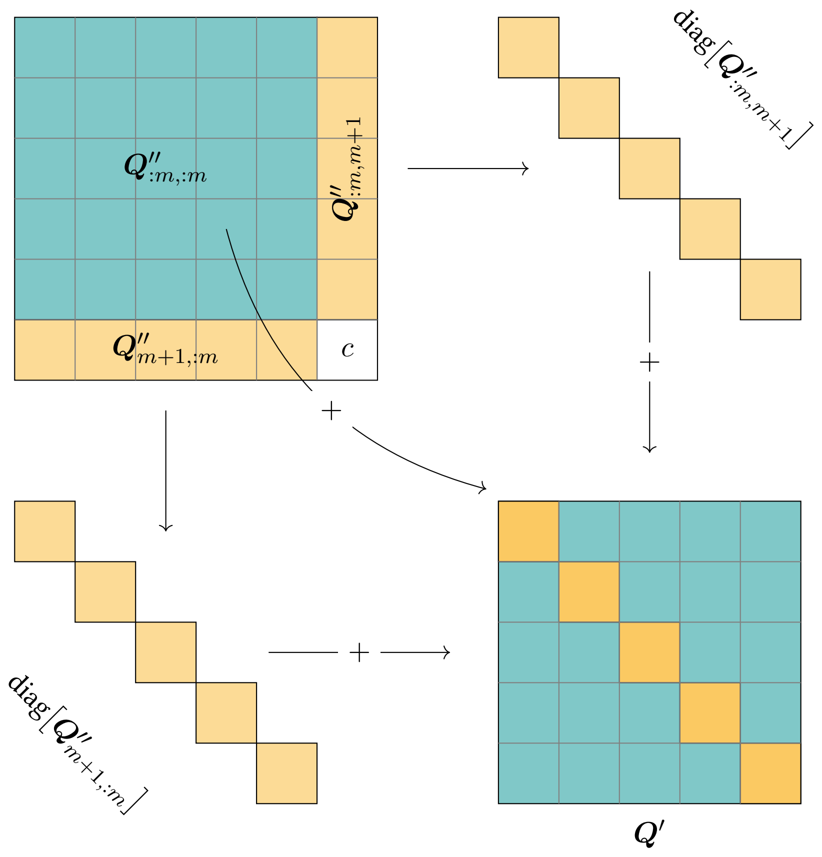


Figure 8.3.: Computation of Q' from Q''. Source: [3].

(III) $\forall i' \neq i, j' \neq j$, clamp $z_{\iota(i',j,k)} = 0$ and $z_{\iota(i,j',k)} = 0$
(forbid the same value within the same row or column)

(IV) For all cells (i'', j'') in the same block as (i, j), clamp $z_{\iota(i'',j'',k)} = 0$ for all
(forbid the same value within the same block)

In principle, only operation (I) is strictly required, but operations (II) to (IV) reduce the size of the QUBO instance further by exploiting domain knowledge. Using only the operations (I) and (II), the QUBO size is reduced by $9|C|$.

8.3. A Practical Example

As a benchmark puzzle, we use a hard Sudoku puzzle from the New York Times website[25] shown in Fig. 8.1. 24 clues are given, from which we extract the corresponding set $C =$

[25] https://www.nytimes.com/puzzles/sudoku/hard (puzzle from 8 January 2024, last accessed June 3, 2025)

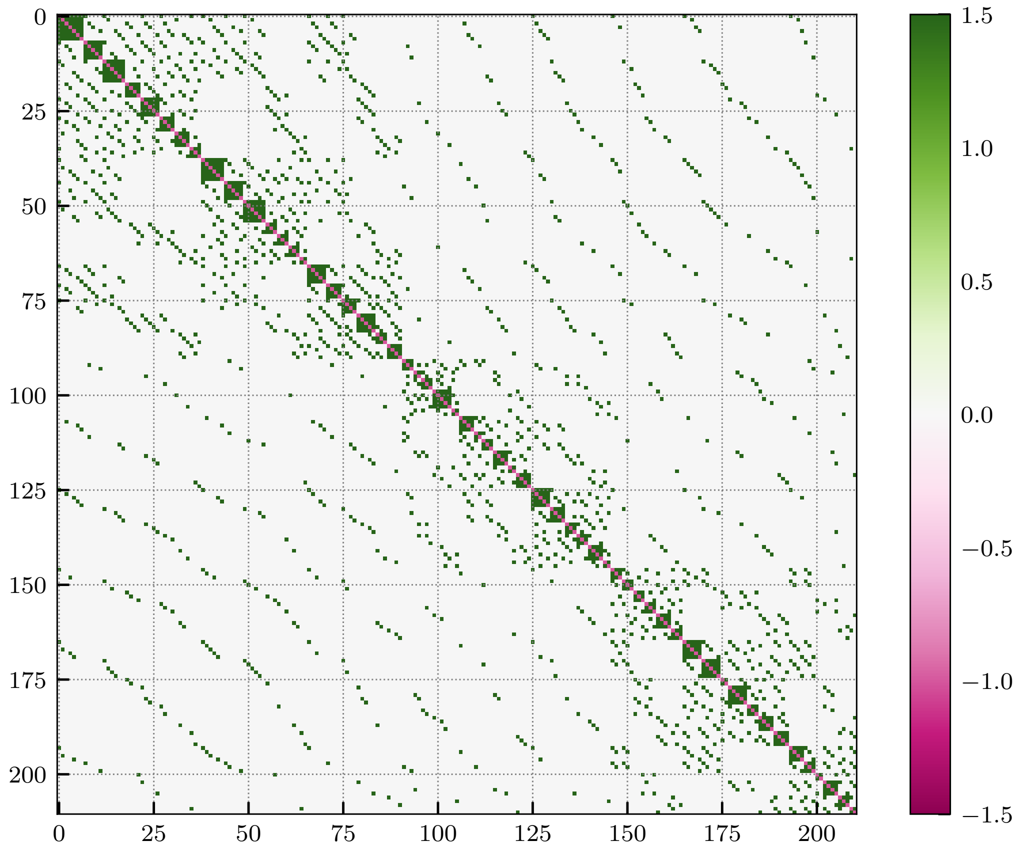


Figure 8.4.: QUBO parameter matrix for the hard Sudoku puzzle of the day by the New York Times on January 8, 2024. The image shows the symmetric matrix $0.5(\boldsymbol{Q}' + \boldsymbol{Q}'^{\intercal})$ after applying all clamping operations (I) to (IV); the resulting optimization problem has 211 variables. Source: [3].

$\{(1,8,2),(2,5,9),\ldots,(9,7,5)\}$. By applying operations (I) and (II) we can reduce the number of variables from 729 to 513, and further down to only 211 variables by applying (III) and (IV). The QUBO matrix $\boldsymbol{Q}'$ is shown in Fig. 8.3. Two more variables could be clamped by the QPRO+ preprocessing algorithm [92], which, similar to the techniques shown in Chapter 6, uses bounds to determine variable assignments that are necessarily optimal. This step is optional. The final QUBO instance with $n = 209$ variables was solved using SA [179] and QA, as we have already seen in Chapter 7.

D-Wave implementations were used for both methods: `SimulatedAnnealingSampler` from the `dwave-neal` package for SA, and `DWaveSampler` from `dwave-system` for QA. Both are set to perform 1000 readouts, all other parameters were left on their default settings to be set heuristically.

As a correct solution to this QUBO problem contains exactly 81 numbers, each giving a reward of -1, and no constraints are violated, we know by construction that the minimal energy of $\boldsymbol{S}$ is -81, allowing us to easily verify if a given solution $\boldsymbol{z}$ is correct by checking if $f_{\boldsymbol{S}}(\boldsymbol{z}) = \boldsymbol{z}^{\intercal}\boldsymbol{S}\boldsymbol{z} = -81$. This property does not change through clamping, as the operations (I) to (IV) never introduce constraint violations, assuming C itself is valid.

The results, as in the previous chapter, reflect the challenges of QA hardware: SA is able to find the optimal solution quite reliably within 1000 readouts with an average energy of -75.047 ± 1.822 per solution. QA on the other hand was *not* able to find the correct solution, even after repeating the experiment 15 times with 1000 readouts each. The average energy of each shot is around -53.795 ± 4.321, which is quite far from the theoretical minimum of -81. Contributing factors are probably *(i)* the restricted topology of D-Wave's QA devices, leading to variable duplication, and increasing the overall problem complexity as a consequence (cf. Section 2.4.3), as well as *(ii)* a lack of fine-grained manual hyperparameter tuning (e.g., manually adjusting chain strength and qubit mapping), which exceeded the scope of this evaluation.

8.4. Effect of Puzzle Difficulty

In a final experiment we want to investigate how a puzzle's difficulty correlates with the performance of solvers on the corresponding QUBO instances. To obtain a range of Sudoku puzzles of varying difficulty, we use the data set "3 million Sudoku puzzles with ratings" by David Radcliffe, available on Kaggle[26]. It contains 3 million Sudoku games with varying number of clues. As a rule of thumb, the hardness of a Sudoku puzzle negatively correlates with the number of clues it has. The data set contains puzzles with the number of clues n_c ranging from 19 to 31. For each value of n_c, we choose the first puzzle in the data set with that number of clues, leaving us with 13 puzzles which we, once again, solve with both SA and QA. This time, we perform 2000 readouts (which is the maximum number of readouts the QA device allows) and record the energies of all samples. The results are visualized as violin plots in Fig. 8.5, showing the mean, minimum, and maximum over all energy values, along with their relative distributions.

Once again, SA outperforms QA, finding the correct solution for 7 out of 13 puzzles within 2000 samples, the hardest puzzle having 23 clues. As expected, puzzles with more clues are generally easier to solve for both methods, as the number of variables decreases with a higher number of clues. Below a certain number of clues, either method is able to find the optimal solution within the number of readouts performed. For QA, this threshold is above our initial upper limit of 31: A few more trials reveal that the lowest number of clues for which QA is able to solve puzzles is around 36, and we were able to obtain one correct solution for 35. At this point, the number of binary variables approaches 100, which seems to be a threshold for the current quantum hardware.

[26] `https://www.kaggle.com/datasets/radcliffe/3-million-sudoku-puzzles-with-ratings` (last accessed June 3, 2025)

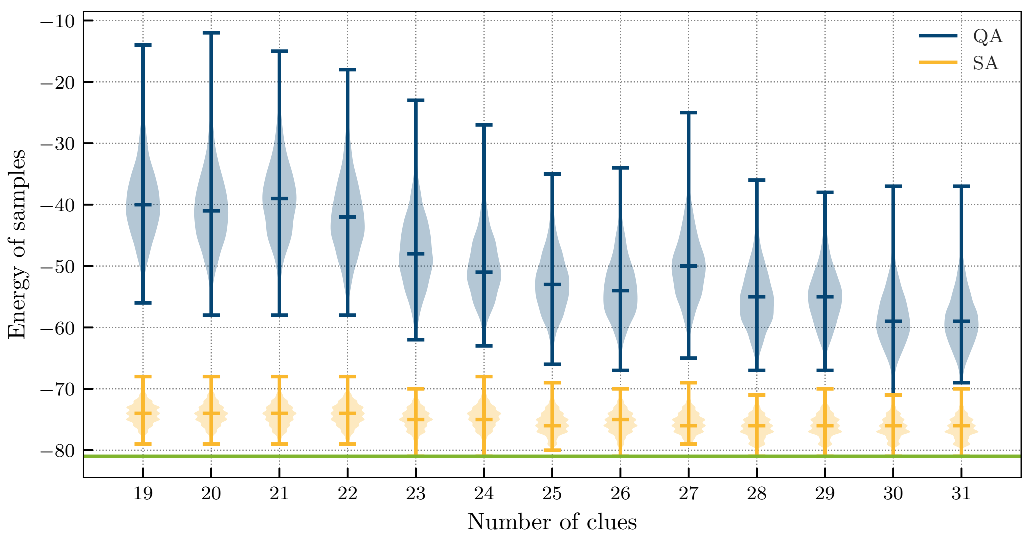


Figure 8.5.: Sample energy distribution for 13 Sudoku puzzles of varying number of clues, using Simulated Annealing and Quantum Annealing with 2000 readouts each. The violin plots show minimum and maximum values as well as medians; lower energy is better. The green line is the lowest possible energy of -81.

8.5. Sampling Sudoku Puzzles from Quantum States

The primary aim of the QUBO formulation is to *solve* Sudoku puzzles. However, it can also be used to create new puzzles: As the un-clamped weight matrix $\boldsymbol{S}$ from Eq. (8.2) has energy -81 for all valid solved Sudoku puzzles, all of them are equally likely under the respective Gibbs distribution $p_{\boldsymbol{S}}(\boldsymbol{z}) = \exp\left[-\beta f_{\boldsymbol{S}}(\boldsymbol{z}) - A(\boldsymbol{S},\beta)\right]$, where β is the inverse system temperature and $A(\boldsymbol{S},\beta) = \log \sum_{\boldsymbol{z}'\in\mathbb{B}^n} \exp\left[-\beta f_{\boldsymbol{S}}(\boldsymbol{z}')\right]$ the log partition function. This means that for $\lim_{\beta\to\text{inf}}$, the probabilities for all $\boldsymbol{z}' \in \mathbb{B}^n$ with $f_{\boldsymbol{S}}(\boldsymbol{z}') > -81$ approaches 0, leaving a uniform distribution over all solved Sudoku puzzles. On real quantum devices, and on NISQ devices in particular, tiny fluctuations of the parameter values, such as those caused by ICE, may bias the distribution, leading to a small subset or even just a single solution dominating the sampler output (cf. [180]). Additionally, embedding $\boldsymbol{S}$ with $n = 729$ onto the D-Wave annealer's qubit topology is not possible.

While true sampling from a quantum Gibbs distribution is still not feasible, we can instead use SA with a random initialization to simulate random sampling, circumventing these problems for now. To this end we ran SA on $\boldsymbol{S}$, performing $10,000$ readouts. 221 of the samples had an energy value of -81, meaning they are valid Sudoku solutions, which we further confirmed to be pairwise distinct from each other.

Algorithm 8.1 Algorithm for sampling Sudoku puzzles

```
Input:   $K, n_c \in \mathbb{N}, 17 \leq n_c < 81$
Output:  Valid Sudoku puzzle
loop
    $\boldsymbol{s} \leftarrow$ sample from $\boldsymbol{S}$ with $f_{\boldsymbol{S}}(\boldsymbol{s}) = -81$
    $C \leftarrow$ generate $n_c$ clues from $\boldsymbol{s}$
    $\boldsymbol{S}'_C \leftarrow$ clamp $\boldsymbol{S}$ with $C$
    $O = \emptyset$
    for all $k = 1, \dots, K$ do
        $\boldsymbol{s}' \leftarrow$ sample from $\boldsymbol{S}'_C$ with $f_{\boldsymbol{S}'_C}(\boldsymbol{s}') = -81$
        $O \leftarrow O \cup \{s'\}$
    end for
    if $|O| = 1$ then
        return $C$
    end if
end loop
```

8.5.1. Generating Unambiguous Puzzles

Generating solved Sudoku grids this way is relatively easy. However, generating *good* Sudoku puzzles and their initial set of clues is more challenging: Simply choosing a random subset of squares and deleting all remaining numbers leads to ambiguous puzzles, meaning that more than one solution is valid, which is generally undesirable. The smallest number of clues necessary for an unambiguous Sudoku puzzle was proven to be 17 [181]. With this knowledge, an algorithm to sample Sudoku puzzles is sketched in Algorithm 8.1.

It is based on the idea that we can sample a solved Sudoku puzzle, generate a random set of clues, and then repeatedly sample valid solutions from the clamped QUBO instance. If after K tries only a single solution was found, we conclude that the clamped QUBO instance has a unique solution, and, consequently, the Sudoku puzzle defined by C is unambiguous. Obviously, this algorithm provides no theoretical guarantees: The larger we choose K, the more confident we can be that C has a unique solution, however, ambiguous puzzles can still occur by chance. In addition, if we choose n_c close to 17, this procedure becomes increasingly unreliable, as the search space gets larger due to the higher number of variables in $\boldsymbol{S}'_C$, while the number of unambiguous puzzles gets smaller.

8.6. Concluding Remarks

This chapter presented a strategy to encode Sudoku puzzles as QUBO instances and solve them using QA. For any given puzzle, the initial clues can be incorporated into the QUBO weight matrix by clamping, an alternative method of enforcing constraints without resorting to extensive use of penalty weights. By using this method, the number of binary variables dictating the search space size is reduced considerably, and the ratio of valid to invalid bit strings is increased. The property that the resulting QUBO problems are smaller for larger clue sets C aligns nicely with the intuition that Sudokus are simpler the more clues are given. We saw that this method solves Sudoku successfully, both using a classical SA solver and a quantum annealer. While SA could solve all presented test puzzles, QA works only for rather easy puzzles with around 36 clues at this point in time, mainly due to NISQ constraints.

Lastly, we showed that QA and (to a certain extent) SA can be used for sampling solved Sudoku grids, and devised an algorithm for generating new Sudoku puzzles. It produces puzzles with a unique solution with a certain probability that can be controlled by a hyperparameter K.

In conclusion, this chapter serves as an example of a NISQ-friendly approach to solving a problem with QC by trying to keep the problem size small and avoiding unnecessary constraints, which may both increase the DR and lead to an energy landscape that is easier to traverse for heuristic solvers and quantum devices alike.

9. Predicting Process Stability in Machining

In Chapters 7 and 8 we have seen two applications of AQC for solving optimization problems. For this last application chapter, we once again turn to the paradigm of GQC to approach an ML problem. While the previous problems arose in the context of games and puzzles, the focus of this chapter is a problem of much more immediate relevance for practical applications: Predicting the stability of milling processes in the context of machining.

This chapter is based on publication [8]. The author of this thesis constructed and pre-trained the custom feature map, extended the RQSVM model to perform regression, designed and conducted all experiments using the provided machining data, created the plots, and wrote most of the paper.

Milling is a manufacturing process that is critical for a wide range of mechanical engineering applications. During this process, a rotary cutter removes material from a workpiece, producing high-precision products and parts that are used in numerous industries, including aerospace [182], the automotive industry [183], and in medical applications [184]. The stability of milling processes is a central factor, impacting the quality and efficiency of production. While theoretical approaches exist for modeling stability, these methods are computationally intensive due to the complex dynamics involved. To circumvent these challenges, we use an ML approach to approximate stability predictions.

As the ML model of choice, we use the SVM, whose QC variations were discussed extensively in Chapter 4. In this chapter, we adapt the RQSVM model presented in Section 4.2. As it is originally a classifier, we extend it to a regression model in Section 9.2 by swapping the underlying SVM model and interpreting the measurements of the resulting quantum state differently.

Real-world applications, particularly at industrial scale, often present big challenges to both classical ML and contemporary QC in particular: They frequently deal with large amounts of data, a large number of variables or features, and they produce noisy and incomplete data. The RQSVM model uses only $\mathcal{O}(\log_2(d))$ qubits for d-dimensional input data, making efficient use of contemporary quantum resources and adapting well to the problem domain at hand.

Using a quantum model for stability prediction in machining opens up the possibility of future exploitations of quantum feature maps [185]. Such functions, mapping classical data into a Hilbert space, may be better suited than classical approaches for capturing the underlying physical effects causing the stability behavior observed in practice.

9.1. Stability of Machining Processes

Milling is a fundamental machining process used to shape solid materials by removing material from a workpiece through the rotational motion of a cutting tool. The process typically involves a multi-toothed cutting tool called a *milling cutter*, which rotates at high speeds and interacts with a stationary or moving workpiece to produce the desired shape. Milling machines are highly versatile, capable of creating complex geometries, making them essential in manufacturing.

It plays a critical role in numerous industries due to its ability to produce precise, high-quality parts: In the aerospace industry, milling is used to manufacture lightweight and complex components such as turbine blades and other important structural parts [182]. In the automative industry, many engine components are produced this way [183]. Similarly, parts for wind turbines, gas turbines, and other energy systems often rely on milling for precise manufacturing [186, 187]. It further enables the creation of surgical instruments and implants, requiring both high precision and intricate detailing [184].

One of the major challenges in milling is ensuring process stability. A stable milling process results in smooth, consistent cuts without undesired oscillations or vibrations. When the process becomes unstable, it can result in a phenomenon known as *chatter*, a self-excited vibration that can have negative effects, including *(i)* reduced surface quality, potentially rendering the part unsuitable for its intended application, *(ii)* increased tool wear and breakage due to excessive forces generated by chatter, and *(iii)* decreased efficiency, as unstable milling may require reduced cutting speeds and depths, lowering productivity and increasing operational costs [188]. To predict and avoid instability, engineers rely on stability lobe diagrams, indicating conditions under which the process is stable. However, generating these diagrams through physical simulations is computationally intensive, as it requires modeling the interactions between the tool, workpiece, and machine dynamics [189, 190].

ML offers an alternative, where these complex factors are modeled implicitly by training a model of the stability behavior from measurements [191, 192], which can even be transferred between different machine tools [193]. In Section 9.2, we discuss a similar strategy, but using an RQSVM model that can be deployed on a gate-based quantum computer.

9.1.1. Data Set of Stability Limits

For later evaluation, we use a data set from [193], containing empirically determined stability limits $a_{e,\lim}$ of milling processes, the input features being spindle speed n and tool

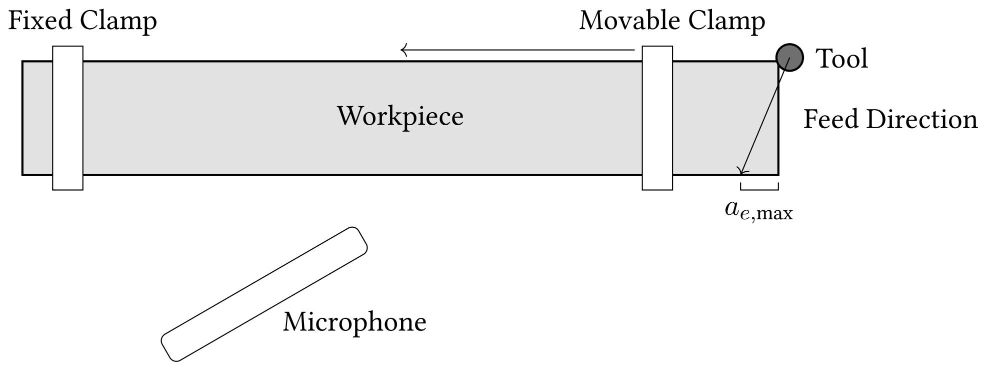


Figure 9.1.: Schematic top-down view of the data collection setup for the stability limit data set. Adapted from [8].

wear w[27]. The data collection setup is shown in Fig. 9.1: A workpiece was fixed with a clamp, and a milling tool performed side milling, where the radial cutting depth a_e was linearly increased to a maximum value $a_{e,\text{max}}$. This process was repeated with spindle speeds varied from 4000 to 8000 RPM with 50 RPM increment. All other parameters were kept fixed. The wear state of the currently used tool was recorded as a value proportional to the volume of material removed per unit length of the tool. During the milling process, a microphone recorded sounds, from which the point at which the process became unstable (i.e., the radial cutting depth $a_{e,\text{lim}}$) is determined. If the process did not become unstable, i.e., $a_{e,\text{lim}} > a_{e,\text{max}}$, we drop the corresponding data point.

To obtain a more varied data set, these tests were carried out on two different machining centers, DMU 50 and DMU 50 eVolution, which we name DMU$_A$ and DMU$_B$ in the following. Additionally, multiple milling tools were used throughout the experiments, yielding multiple measurements per wear state w. More precisely, six tools were used, three each for DMU$_A$ and DMU$_B$, labeled $T_A^{(1)}$, $T_A^{(2)}$, $T_A^{(3)}$, and $T_B^{(1)}$, $T_B^{(2)}$, $T_B^{(3)}$, accordingly. The measurements are collected separately for each DMU, resulting in two data sets $\mathcal{D}_A$ and $\mathcal{D}_B$, containing $|\mathcal{D}_A| = 895$ and $|\mathcal{D}_B| = 829$ points. All data points are shown in the top halfs of Figs. 9.3 and 9.4 for $\mathcal{D}_A$ and $\mathcal{D}_B$, respectively. For a more detailed description of this data set, its collection setup, and the remaining process parameters, refer to [193, 8].

9.2. Applying Quantum Support Vector Regression

In this section, we apply predict the stability limit of milling processes using a quantum model. To this end, we device a custom feature map to allow for more complex, non-linear prediction functions. As our target variable $a_{e,\text{lim}}$ is continuous, we use an extension of

[27]Note that n is used this way in machining literature. We use it to denote the spindle speed only within this chapter, as it may otherwise be confused with the number of variables, e.g., of a Qubo problem. This conflicting usage is avoided throughout this chapter.

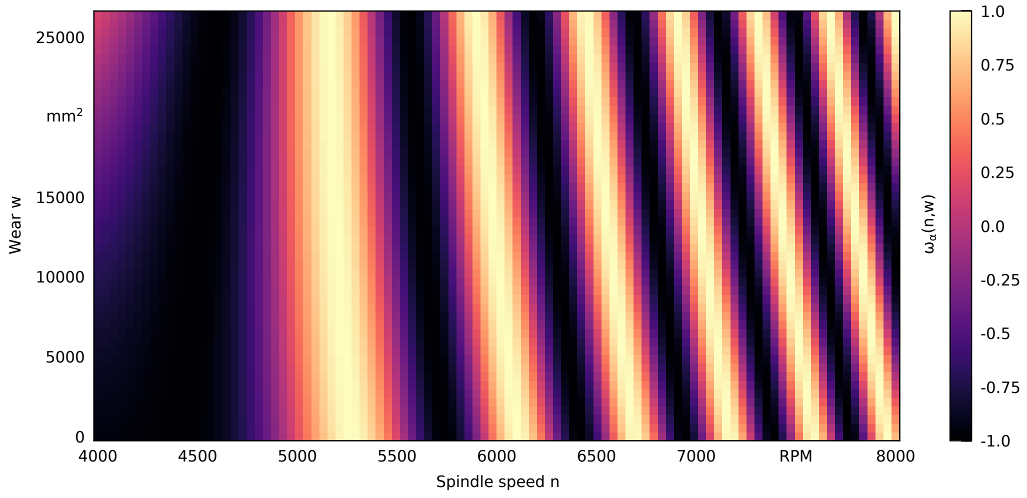


Figure 9.2.: Exemplary function $\omega_{\boldsymbol{\alpha}}$. Source: [8].

the RQSVM model discussed in Section 4.2, resulting in an ϵ-Real-part Quantum Support Vector Regressor (RQSVR) model capable of performing regression.

9.2.1. Building a Feature Map

As we see in Figs. 9.3 and 9.4 (top), the stability data exhibits a wave-like pattern. To capture this behavior, a custom feature map is devised to augment the collected data. To this end, a wave function $\omega_{\boldsymbol{\alpha}}$ is defined with

$$\omega_{\boldsymbol{\alpha}}(n, w) = \cos\left(\alpha_1 + \alpha_2 n + \alpha_3 w + \alpha_4 n^2 + \alpha_5 n w + \alpha_6 w^2\right),$$

where $\boldsymbol{\alpha} \in \mathbb{R}^6$ parametrizes the wave's frequency by a 2^{nd} degree polynomial over n and w. The function is shown in Fig. 9.2 for a fixed value of $\boldsymbol{\alpha}$.

Using this wave component, a final feature map $\boldsymbol{\phi}_{\cos}$ is constructed as

$$\boldsymbol{\phi}_{\cos}(n, w; \boldsymbol{\alpha}) = \left(n, w, n^2, nw, w^2, \omega, n\omega, w\omega, n^2\omega, nw\omega, w^2\omega\right)^\mathsf{T},$$

with ω being short for $\omega_{\boldsymbol{\alpha}}(n, w)$. To obtain fitting values for $\boldsymbol{\alpha}$, a function

$$f(n, w; \boldsymbol{\alpha}, \boldsymbol{\beta}, c) = \boldsymbol{\beta}^\mathsf{T} \boldsymbol{\phi}_{\cos}(n, w; \boldsymbol{\alpha}) + c \tag{9.1}$$

was defined and used as a prediction function to perform a least-squares fit on $\mathcal{D}_A$ and $\mathcal{D}_B$ separately by adapting the parameters $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and c. First, each data column was normalized to the unit interval, and the L-BFGS-B algorithm [194] was applied to minimize the MSE between $f(n, w; \boldsymbol{\alpha}, \boldsymbol{\beta}, c)$ and $a_{e,\text{lim}}$, for which the Python package `scipy` was used[28]. As the optimization proved to be difficult, multiple L-BFGS-B runs with different

[28]`https://scipy.org` (last accessed June 3, 2025)

Table 9.1.: Parameters $\boldsymbol{\alpha}$ found for $\mathcal{D}_A$ and $\mathcal{D}_B$, and the resulting MSE. Source: [8].

Parameter	$\mathcal{D}_A$	$\mathcal{D}_B$
α_1	15.52749483	-2.87183284
α_2	-4.46971848	11.95336368
α_3	-9.55499409	0.99781459
α_4	-14.22448621	28.58595761
α_5	-2.38118747	2.27955561
α_6	14.75707122	-4.34204599
MSE	0.12337479	0.21384462

initializations was performed. To this end, a simple local search over the initial parameters was performed by sampling $\boldsymbol{\alpha}_i^{(0)}, \boldsymbol{\epsilon}_i^{(t)} \sim \mathcal{N}(0, 10)$ i.i.d. for all i and t, and setting $\boldsymbol{\alpha}^{(t+1)} = \boldsymbol{\alpha}_{\text{min}}^{(t)} + \boldsymbol{\epsilon}^{(t)}$, where $\boldsymbol{\alpha}_{\text{min}}^{(t)}$ yielded the lowest MSE across all iterations $t' \leq t$. This procedure is, in essence, a $(1+1)$ EA used as a hyperparameter search (cf. Section 2.2.2). The termination criterion was reached when no improvement was found after 1000 iterations.

The final values of $\boldsymbol{\alpha}$ found for each data set are listed in Table 9.1, along with their resulting MSE values. These values were used to generate two new data sets $\mathcal{F}_A$ and $\mathcal{F}_B$ containing the features $\boldsymbol{\phi}_{\cos}(\tilde{n}, \tilde{w}; \boldsymbol{\alpha}^*)$, where $\tilde{n}$ and $\tilde{w}$ are the normalized original features, and $\boldsymbol{\alpha}^*$ is the optimal value of $\boldsymbol{\alpha}$ found for the respective data set, $\mathcal{D}_A$ and $\mathcal{D}_B$.

9.2.2. Support Vector Regression

SVM models are, by default, binary classifiers, as their model function only outputs $+1$ or -1 (see Eq. (2.11)). However, the ϵ-Support Vector Regressor (SVR) [60, 113] is a variation of the original SVM with a modified training procedure and model function, capable of predicting real values.

Definition 9.1 (Primal ϵ-SVR [113]). *Given a labeled data set $\mathcal{D} = \{(\boldsymbol{x}^i, y_i)\}_{i \in \{1,\dots,N\}} \subset \mathbb{R}^d \times \mathbb{R}$ and a feature map $\varphi : \mathbb{R}^d \to \mathbb{R}^f$, the* primal ϵ-SVR *is the optimization problem*

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\xi}'} \; & \frac{1}{2}\|\boldsymbol{w}\|_2^2 + C\mathbf{1}^\intercal\boldsymbol{\xi} + C\mathbf{1}^\intercal\boldsymbol{\xi}' \\ \textit{s.t. } & \boldsymbol{w}^\intercal\varphi(\boldsymbol{x}^i) - b - y_i \geq \epsilon + \xi_i, \\ & y_i - \boldsymbol{w}^\intercal\varphi(\boldsymbol{x}^i) + b \geq \epsilon + \xi_i', \\ & \xi_i, \xi_i' \geq 0 \; \forall \, i \in \{1, \dots, N\}, \end{aligned}$$

where $\epsilon, C > 0$ are hyperparameters.

In contrast to Def. 2.11, this optimization problem has two hyperparameters: The parameter ϵ defines a margin of values which are considered *correct*, i.e., if the prediction is less than ϵ away from the true value y, no penalty is added. If in fact the error is larger than ϵ, the slack values $\boldsymbol{\xi}$ or $\boldsymbol{\xi}'$ are increased, depending on if the prediction overshoots or undershoots the true value. Hyperparameter C controls the penalization of wrong predictions.

Just like the primal SVM, the ϵ-SVR has a dual formulation, which can be derived and solved by means of Lagrangian optimization (see [113] for details). The final prediction function of the linear ϵ-SVR is given by

$$\hat{y}_{\boldsymbol{w},b}(\boldsymbol{x}) = \boldsymbol{w}^\intercal \varphi(\boldsymbol{x}) - b.$$

As we see, it is nearly identical to Eq. (2.11), differing only in the missing sign function, which yields a real-valued output.

The RQSVM can be extended seamlessly to perform ϵ-SVR regression. To this end, we simply swap out the parameter training procedure, using LIBSVM [113] to solve the problem given in Def. 9.1 and obtain the optimal weights $\boldsymbol{w}$, which are then embedded into the quantum circuit as described in Section 4.2. The measurements of the final quantum state are evaluated identically, as well, only omitting the sign function at the end.

9.3. Evaluation on Stability Data

Combining the findings of the preceding sections, an ϵ-RQSVR model is trained on the enhanced data sets $\mathcal{F}_A$ and $\mathcal{F}_B$. The error margin was fixed to $\epsilon = 0.1$. To determine hyperparameter C, a range of values at different orders of magnitude were tested ($C \in \{10^{-3}, 10^{-2}, \dots, C^2\}$). For each, a 10-fold cross validation was performed. To this end, the data points were shuffled, split into 10 subsets of roughly the same size, and the MSE recorded for each subset after training an ϵ-RQSVR on the remaining 9 subsets. The mean of all 10 MSE yields the cross-validated MSE. It was found that $C = 1$ leads to the best performance, with a mean MSE of 0.15470591 for $\mathcal{F}_A$ and 0.25276193 on $\mathcal{F}_B$.

Figures 9.3 and 9.4 (bottom) show the training data predictions, obtained by using $\mathcal{F}_A$ and $\mathcal{F}_B$ as inputs to their respective ϵ-RQSVR models. For each data point, 10^5 measurements of the simulated quantum state were taken to compute the prediction. Figures 9.5 and 9.6 show the prediction function for different values of tool wear w fixed at regular intervals between 0 and 256 cm^2. Additionally, the uncertainty introduced by quantum measurements is shown as an area around the plot, denoting one standard deviation.

All figures demonstrate that the feature map captures the wavy pattern of the stability data. Particularly for $\mathcal{F}_B$, the prediction is visually very similar to ground truth, exhibiting the same characteristic wave-like pattern. For $\mathcal{F}_A$, the cosine component of the feature map is used to a much lesser degree, leading to a much *straighter* prediction function, as seen in Fig. 9.5. Due to the complexity of the highly non-linear optimization problem

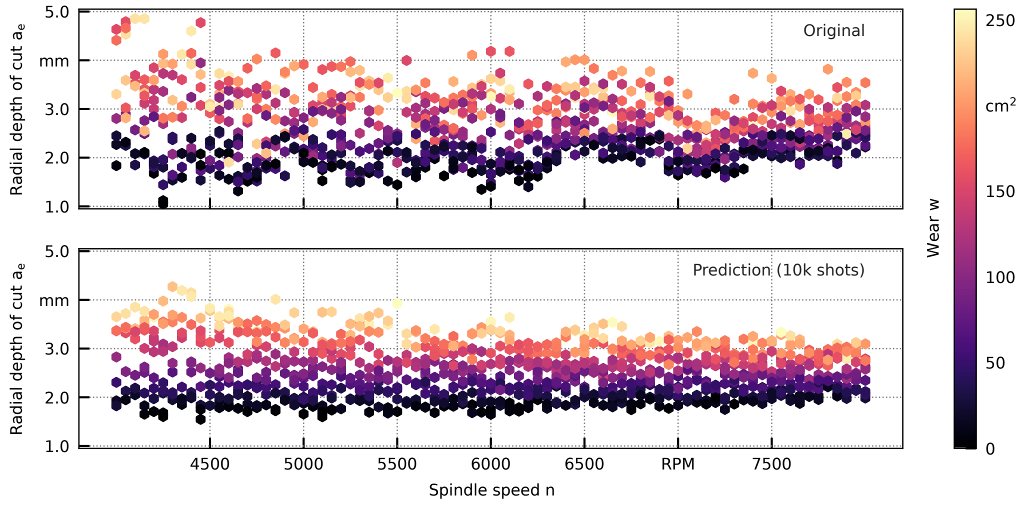


Figure 9.3.: Top: Data set $\mathcal{D}_A$. **Bottom:** Prediction of RQSVR model trained on $\mathcal{F}_A$. Source: [8].

in Section 9.2.1, it is possible that a better solution exists, but was not found using the iterative procedure. In Fig. 9.4, we see that the model predicts another rise of the stability limit at around 7700 RPM, which is not present in the original data, though. This implies that the feature map may not be expressive enough to model this particular part of the training data. For lower values of n, however, the predictions generally align closely with ground-truth.

9.3.1. Predicting Tools

The milling tools $T_A^{(1)}, \ldots, T_B^{(3)}$ are of the same make, and should—in theory—behave the same under the same conditions. In reality, however, slight manufacturing imperfections causes each individual tool to behave differently in detail. In a final experiment, this variation among milling tools is investigated.

As mentioned in Section 9.1.1, each data set is comprised of measurements taken from three tools labeled $T_\cdot^{(1)}$, $T_\cdot^{(2)}$, and $T_\cdot^{(3)}$, for each of the two DMUs. In other words, we can describe the data sets as $\mathcal{D}_\cdot = \bigcup_{i=1}^3 \mathcal{T}_\cdot^{(i)}$, where $\mathcal{T}_\cdot^{(i)}$ contains all features recorded using tool $T_\cdot^{(i)}$. Intuitively, the tools should behave the same if we can train a model on two tools and accurately predict the behavior of the third tool. To test this theory, we perform the following steps for every DMU $M \in \{A, B\}$ and tool $i \in \{1, \ldots, 3\}$:

1. Split the data set into

$$\mathcal{D}_M^{\text{train}} = \bigcup_{\substack{j=1,\\ j\neq i}}^{3} \mathcal{D}_M^{(j)}, \qquad \mathcal{D}_M^{\text{test}} = \mathcal{D}_M^{(i)}.$$

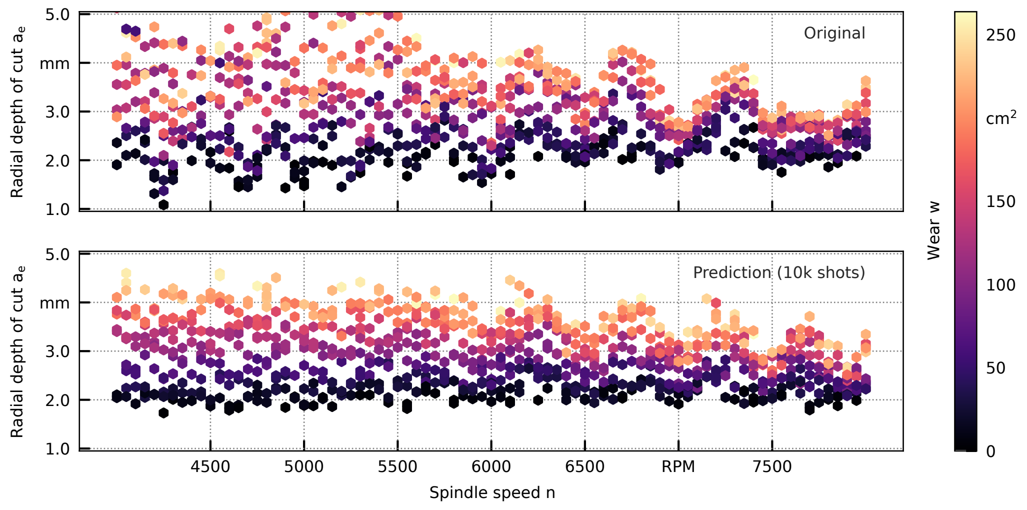


Figure 9.4.: Top: Data set $\mathcal{D}_B$. **Bottom:** Prediction of RQSVR model trained on $\mathcal{F}_B$. Source: [8].

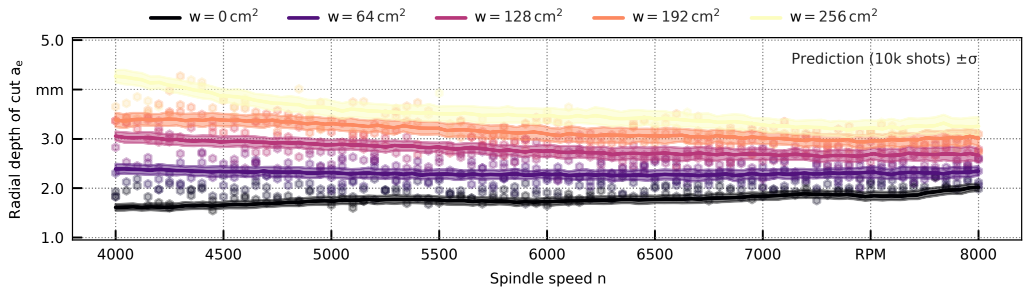


Figure 9.5.: Model prediction for $\mathcal{D}_A$ for varying fixed wear values, with uncertainty over measurement noise (one standard deviation). Source: [8].

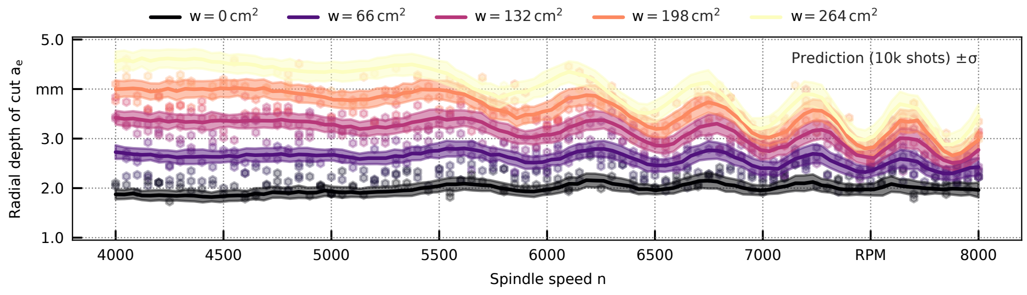


Figure 9.6.: Model prediction for $\mathcal{D}_B$ for varying fixed wear values, with uncertainty over measurement noise (one standard deviation). Source: [8].

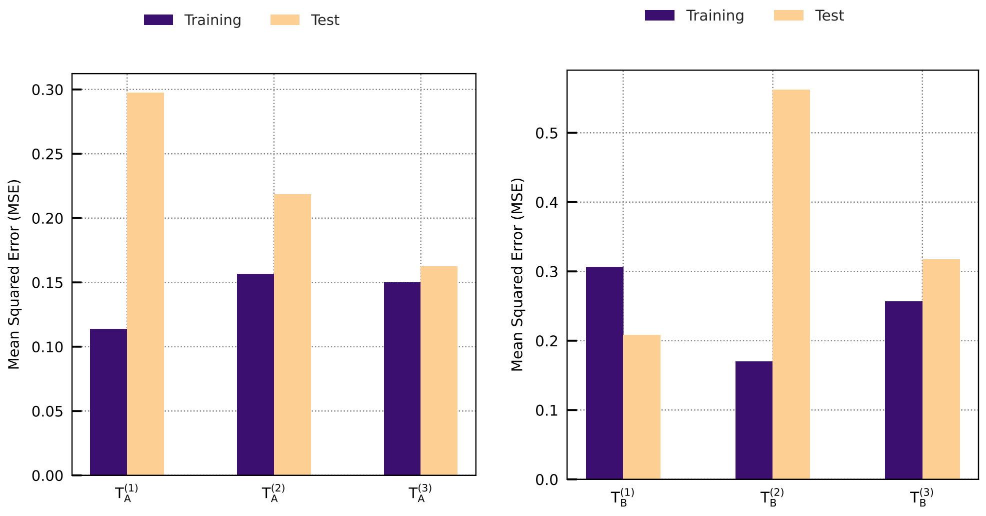


Figure 9.7.: MSE of RQSVR models trained on different tool subsets. The stability of the tool on the x-axis is predicted after training the model on the remaining two tools. **Left:** Tools from $\mathcal{D}_A$. **Right:** Tools from $\mathcal{D}_B$. Source: [8].

2. Construct a feature map $\phi_{\cos}(\cdot;\boldsymbol{\alpha})$ from $\mathcal{D}_M^{\text{train}}$, using the methodology described in Section 9.2.1.
3. Use $\phi_{\cos}(\cdot;\boldsymbol{\alpha})$ to compute $\mathcal{F}_M^{\text{train}}$ and $\mathcal{F}_M^{\text{test}}$.
4. Train an ϵ-RQSVR with $\epsilon = 0.1$ and $C = 1$ on $\mathcal{F}_M^{\text{train}}$.
5. Predict $\mathcal{F}_M^{\text{train}}$ and $\mathcal{F}_M^{\text{test}}$ using the trained model and record the MSE values.

The results of this experiment are shown in Fig. 9.7 as bar plots. If the tools were perfectly identical, we would expect that all MSE values are approximately equal across all tools. However, we find that the models' performance is different for some tools. Notably, both data sets seem to contain one tool in particular that is more difficult to predict from the remaining tools: Both $T_A^{(1)}$ and $T_B^{(2)}$ yield the highest test MSE among all tools, being roughly twice as high as for the remaining tools, while simultaneously having the lowest training MSE, implying that the remaining tools' behavior is more similar to each other. The fact that the training MSE is higher for the remaining tools further reinforces this theory, as the inclusion of features from the outlier tool seems to lower the overall prediction quality rather than improve it. This is particularly noteworthy for $T_B^{(1)}$, where the training error is higher than the test error, which is unexpected, as the performance of any ML model is typically worse on unseen data (cf. [9, Sec. 7.4]). A possible explanation of this atypical behavior is that $T_B^{(2)}$ is so difficult to predict that its inclusion in the training MSE of $T_B^{(1)}$ makes it larger than the test error.

In summary, we find the initial hypothesis confirmed that the tools do not behave identically in practice, and some tools are harder to predict than others. It must be noted that this can be due to *(i)* different physical properties of the tools, or *(ii)* a different feature map quality resulting from training on the other tools, though the second point correlates with the first one.

9.4. Concluding Remarks

This chapter has discussed a third application of QC to a real-life problem, namely to the prediction of stability limits in milling processes. To this end, a custom feature map based on domain knowledge was constructed, and an ϵ-RQSVR trained on the resulting features, which is an extension of the RQSVM presented in Section 4.2. Experiments showed that this model is able to predict the stability limits contained in the empirical data set accurately, while introducing some noise caused by quantum measurements. In another experiment, the similarity of the different tools used during data collection was analyzed by trying to predict their stability behavior from all other tools. The results show that the tools, despite being of the same make, behave differently and cannot be perfectly predicted from each other. Possibly, a transfer learning approach similar to [193] could be employed for this scenario, in order to fine-tune models between different tools.

Similar to other practical applications, the advantage of using QC for predicting milling stability is, at this point, of theoretical nature. While the feature map in Section 9.2.1 relies on domain knowledge and is limited in its expressivity, quantum computers beyond the NISQ era may be capable of performing both feature extraction *and* regression, exploiting the exponentially large Hilbert space of quantum states to encode features that may be even better suited than classical features to predicting the stability limits of milling processes [185], which are themselves a result of the complex interplay of physical properties, after all.

10. Conclusion and Future Work

This thesis has delved deeply into the interplay of optimization, ML and QC in the NISQ era. We posed the questions of how ML, driven by optimization at its core, can benefit from QC methods, and conversely, how classical methods can improve the limited and imperfect quantum resources we have available today.

Concerning the first question, we showed in Chapter 3 that FS, an important pre-processing step in any ML pipeline, can be solved using a Qubo formulation of the corresponding optimization problem on quantum annealers. It was proven that any desired number of features can be obtained simply by weighing feature importance against redundancy, and we showed in a range of practical experiments that the method works well in practice and is competitive to alternative FS approaches. We then turned our attention to SVMs in Chapter 4, which constitute a theoretically well-founded ML model, and showed that QC can be applied twofold: QA can be used to train SVMs under the assumption of discretized weights, which still perform well compared to traditional SVMs with floating-point weights. Given a trained SVM, we showed how to deploy them using GQC while preserving their theoretical properties.

Concerning the second question, we showed how to improve both GQC and QA using classical techniques: In Chapter 5 we used a custom evolution-based optimization routine to "grow" quantum circuits for optimizing given problem Hamiltonians. This lead to smaller, more intelligent circuit designs that are adapted to the specific quantum hardware and its noise characteristics. In Chapter 6 we identified DR as an important factor for the error-proneness of quantum annealers, and introduced algorithms to reduce it, exploiting theoretical bounds on the minimal energy. We showed empirically that, after applying our DR reduction method, QA showed improved performance by finding the global minimizer of Qubo instances more realiably.

Finally, we explored three practical applications of QC to quite different problems. By utilizing a variety of techniques we apply NISQ-style QC more efficiently. In Chapter 7, we solve the problem of placing light sources resource-efficiently using QA as a component within an ADMM optimization loop. Along the way we discovered an unintuitive connection to the SetCover problem, to which our method is also applicable. Chapter 8 uses another Qubo formulation to solve Sudoku puzzles: Here, we use clamping to significantly reduce the problem size instead of resorting to penalty weights, both saving on quantum resources and reducing DR along the way. Finally, in Chapter 9 we apply our RQSVM devised in Section 4.2 to a real-world data set containing milling process data.

Using a domain-specific feature map, we demonstrate that GQC can be used for accurately predicting the stability limit, facilitating high-quality results in machining.

The main results of this thesis rest on sound theoretical foundations, which helps to pave the road leading out of the NISQ era: Owing to the limitations of contemporary quantum hardware, such as small number of qubits, shallow circuit depths, limited entanglement capability, decoherence and gate noise, not all of the methods presented here could be tested and applied on real devices as thoroughly as we would have liked. The scale of problems to which we *were* able to apply physical QC seems, at times, underwhelming compared to the massive achievements currently seen in areas like computer vision and natural language processing. However, this thesis explores a still quite novel computing paradigm, whose true potential has not yet unfolded and may eventually be applicable at state-of-the-art scales.

The practical implications of noise-free, large-scale QC are enormous, as it allows for asymptotic speedups in core components of many algorithms, as well as efficient analog solving of NP-hard optimization problems like QUBO. The advancements presented in this thesis would be immediately usable with such future hardware and lead to more efficient optimization and ML routines, making a contribution to a transformed ML landscape.

During the course of this thesis, numerous unanswered questions have arisen which leave room for future work. Some of these questions have already been discussed in the closing remarks of each separate chapter, and only the most intriguing questions shall be reiterated here. In Section 4.2 we have seen that the RQSVM circuit computes an inner product that is used for the model prediction. Instead of embedding the data into the circuit as-is, we could think about strategies to construct quantum feature maps that transform the data before computing the inner product, which could potentially lead to richer data representations and better model performance.

Concerning quantum circuit evolution, a natural future direction is to further investigate the potential of different basis gate sets and their effectiveness in an evolutionary optimization setting. Quantum gate computers have native gate sets, as well as a restricted qubit topology that allows for two-qubit gates only between certain pairs. Including these constraints into the evolutionary operators seems to be a natural extension to our method, and would make the resulting circuits even better adapted to the NISQ hardware at hand.

Another very intriguing future research direction is gaining more theoretical insights into QUBO problems $\boldsymbol{Q}$ and their corresponding minimizing sets $S^*(\boldsymbol{Q})$. The relation $\equiv$ defined in Def. 2.7 induces equivalence classes on $\mathcal{Q}_n$ for any $n \in \mathbb{N}$, meaning there is a necessarily finite number of QUBO instances with distinct minimizers. This implies further that each class contains one member whose DR is minimal (or whose minimal energy gap is maximal), which would be the ideal QUBO instance to solve on quantum annealers. Finding these *natural representatives* for each minimizing set is a fascinating research endeavor, which could potentially grant deeper insights into the subject of DR reduction of QUBO instances. Our method presented in Chapter 6 has the flaw of being iterative, changing only one parameter at a time. Finding the matrix $\boldsymbol{A}$ in Eq. (6.3) directly (or, at least, in fewer steps) would lead to much more efficient DR reduction. However, it is currently unknown how to maintain the $\sqsubseteq$ relation while modifying the entire matrix at once.

Concerning our application of the RQSVM to the machining data, it would be interesting to connect theoretical knowledge about the stability behavior of milling processes with a quantum feature map. So far, we have seen that we can define a feature map based on sine waves to approximate the lobes of stability occuring in the empirical data. However, there could be a quantum feature map that captures the behavior more faithfully. This research direction intersects our proposed extension of the RQSVM model mentioned above, potentially leading to more expressive QML models with theoretical properties carried over from their classical counterparts.

Quantum Computing is still in its infancy, and its full-scale practical applicability may still be several decades away. To ensure that any work on it done today still has value in the future, it has to rest on theoretical groundwork, not chase after state-of-the-art performances using poorly-understood heuristics. This thesis has added a few items to the list of meaningful things to do with QC, and taken a step towards better quantum-classical optimization and ML. Yet there is still much fundamental knowledge to be discovered and connections to be made.

Bibliography

[1] Sascha Mücke, Thore Gerlach, and Nico Piatkowski. "Optimum-Preserving QUBO Parameter Compression". In: *Quantum Machine Intelligence* 7 (2025). DOI: 10.1007/s42484-024-00219-3.

[2] Nico Piatkowski and Sascha Mücke. "Real-Part Quantum Support Vector Machines". In: *Proceedings of Machine Learning and Knowledge Discovery in Databases.* Vol. 14948. Lecture Notes in Computer Science. Springer, 2024, pp. 144–160. DOI: 10.1007/978-3-031-70371-3_9.

[3] Sascha Mücke. "A Simple QUBO Formulation of Sudoku". In: *Companion Proceedings of the Genetic and Evolutionary Computation Conference.* ACM, 2024, pp. 1958–1962. DOI: 10.1145/3638530.3664106.

[4] Sascha Mücke and Thore Gerlach. "Efficient Light Source Placement using Quantum Computing". In: *Lernen, Wissen, Daten, Analysen.* Vol. 3630. CEUR Workshop Proceedings. CEUR-WS.org, 2023, pp. 478–491.

[5] Sascha Mücke et al. "Feature selection on quantum computers". In: *Quantum Machine Intelligence* 5.1 (2023), pp. 1–16. DOI: 10.1007/S42484-023-00099-Z.

[6] Lukas Franken et al. "Quantum Circuit Evolution on NISQ Devices". In: *Proceedings of the IEEE Congress on Evolutionary Computation.* IEEE, 2022, pp. 1–8. DOI: 10.1109/CEC55065.2022.9870269.

[7] Sascha Mücke, Nico Piatkowski, and Katharina Morik. "Learning Bit by Bit: Extracting the Essence of Machine Learning". In: *Proceedings of the Conference on "Lernen, Wissen, Daten, Analysen".* Vol. 2454. CEUR Workshop Proceedings. CEUR-WS.org, 2019, pp. 144–155.

[8] Sascha Mücke et al. *Predicting Machining Stability with a Quantum Regression Model.* 2024. arXiv: 2412.04048 [quant-ph].

[9] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition.* Springer Series in Statistics. Springer, 2009. DOI: 10.1007/978-0-387-84858-7.

[10] Sepp Hochreiter. "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions". In: *Int. J. Uncertain. Fuzziness Knowl. Based Syst.* 6.2 (1998), pp. 107–116. DOI: 10.1142/S0218488598000094.

[11] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on Machine Learning.* Omnipress, 2010, pp. 807–814.

[12] Tom B. Brown et al. *Language Models are Few-Shot Learners.* 2020. arXiv: 2005.14165 [cs.CL].

[13] Emma Strubell, Ananya Ganesh, and Andrew McCallum. "Energy and Policy Considerations for Modern Deep Learning Research". In: *Proceedings of the 34th AAAI Conference on Artificial Intelligence.* AAAI Press, 2020, pp. 13693–13696. DOI: 10.1609/AAAI.V34I09.7123.

[14] David Patterson et al. "Carbon Emissions and Large Neural Network Training". In: (2021). arXiv: 2104.10350 [cs.LG].

[15] Peter W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* 26.5 (1997), pp. 1484–1509. DOI: 10.1137/s0097539795293172.

[16] Lov K. Grover. "A Fast Quantum Mechanical Algorithm for Database Search". In: *Proceedings of the 28th Annual ACM Symposium on the Theory of Computing.* ACM, 1996, pp. 212–219. DOI: 10.1145/237814.237866.

[17] Tadashi Kadowaki and Hidetoshi Nishimori. "Quantum annealing in the transverse Ising model". In: *Physical Review E* 58.5 (1998), pp. 53–55.

[18] Yuriy Makhlin, Gerd Schön, and Alexander Shnirman. "Quantum-state engineering with Josephson-junction devices". In: *Reviews of modern physics* 73.2 (2001), p. 357.

[19] Joseph A. Schreier et al. "Suppressing charge noise decoherence in superconducting charge qubits". In: *Physical Review B–Condensed Matter and Materials Physics* 77.18 (2008).

[20] John Preskill. "Quantum Computing in the NISQ era and beyond". In: *Quantum* 2 (2018), p. 79. DOI: 10.22331/q-2018-08-06-79.

[21] Jacob D. Biamonte et al. "Quantum machine learning". In: *Nat.* 549.7671 (2017), pp. 195–202. DOI: 10.1038/NATURE23474.

[22] Jarrod R. McClean et al. "Barren plateaus in quantum neural network training landscapes". In: *Nature Communications* 9.1 (2018). DOI: 10.1038/s41467-018-07090-4.

[23] Kunal Sharma et al. "Trainability of Dissipative Perceptron-Based Quantum Neural Networks". In: *Physical Review Letters* 128.18 (2022). DOI: 10.1103/physrevlett.128.180505.

[24] David Peral García, Juan Cruz-Benito, and Francisco José García-Peñalvo. "Systematic literature review: Quantum machine learning and its applications". In: *Comput. Sci. Rev.* 51 (2024). DOI: 10.1016/J.COSREV.2024.100619.

[25] David H. Wolpert. "The Supervised Learning No-Free-Lunch Theorems". In: *Soft Computing and Industry: Recent Applications.* Springer London, 2002, pp. 25–42. DOI: 10.1007/978-1-4471-0123-9_3.

[26] Seunghun Jin et al. "FPGA Design and Implementation of a Real-Time Stereo Vision System". In: *IEEE Trans. Circuits Syst. Video Technol.* 20.1 (2010), pp. 15–26. DOI: 10.1109/TCSVT.2009.2026831.

[27] B. Angelucci et al. "The FPGA based Trigger and Data Acquisition system for the CERN NA62 experiment". In: *Journal of Instrumentation* 9.01 (2014). DOI: 10.1088/1748-0221/9/01/C01055.

[28] Rym Chéour et al. “Microcontrollers for IoT: Optimizations, Computing Paradigms, and Future Directions”. In: *Proceedings of the 6th IEEE World Forum on Internet of Things.* IEEE, 2020, pp. 1–7. DOI: 10.1109/WF-IOT48130.2020.9221219.

[29] Mykel J. Kochenderfer and Tim A. Wheeler. *Algorithms for Optimization.* The MIT Press, 2019. ISBN: 0262039427.

[30] Thomas Bäck and Hans-Paul Schwefel. “An Overview of Evolutionary Algorithms for Parameter Optimization”. In: *Evol. Comput.* 1.1 (1993), pp. 1–23. DOI: 10.1162/EVCO.1993.1.1.1.

[31] Stefan Droste, Thomas Jansen, and Ingo Wegener. “On the analysis of the (1+1) evolutionary algorithm”. In: *Theor. Comput. Sci.* 276.1-2 (2002), pp. 51–81. DOI: 10.1016/S0304-3975(01)00182-7.

[32] Thomas Jansen and Ingo Wegener. “On the analysis of a dynamic evolutionary algorithm”. In: *J. Discrete Algorithms* 4.1 (2006), pp. 181–199. DOI: 10.1016/J.JDA.2005.01.002.

[33] D. J. Laughhunn. “Quadratic Binary Programming with Application to Capital-Budgeting Problems”. In: *Oper. Res.* 18.3 (1970), pp. 454–461. DOI: 10.1287/OPRE.18.3.454.

[34] Peter L. Hammer and Eliezer Shlifer. “Applications of pseudo-Boolean methods to economic problems”. In: *Theory and decision* 1.3 (1971), pp. 296–308.

[35] Gary Kochenberger et al. “Using the unconstrained quadratic program to model and solve Max 2-SAT problems”. In: *International Journal of Operational Research* 1.1-2 (2005), pp. 89–100.

[36] Florian Neukart et al. “Traffic Flow Optimization Using a Quantum Annealer”. In: *Frontiers ICT* 4 (2017), p. 29. DOI: 10.3389/FICT.2017.00029.

[37] Tobias Stollenwerk, Elisabeth Lobe, and Martin Jung. “Flight Gate Assignment with a Quantum Annealer”. In: *Proceedings of the Workshop on Quantum Technology and Optimization Problems.* Vol. 11413. Lecture Notes in Computer Science. Springer, 2017, pp. 99–110. DOI: 10.1007/978-3-030-14082-3_9.

[38] Christian Bauckhage et al. “Adiabatic Quantum Computing for Kernel k=2 Means Clustering”. In: *Proceedings of the Conference on ”Lernen, Wissen, Daten, Analysen”.* Vol. 2191. CEUR Workshop Proceedings. CEUR-WS.org, 2018, pp. 21–32.

[39] Prasanna Date, Davis Arthur, and Lauren Pusey-Nazzaro. “QUBO formulations for training machine learning models”. In: *Scientific Reports* 11.1 (2021). DOI: 10.1038/s41598-021-89461-4.

[40] Satoshi Matsubara et al. “Ising-Model Optimizer with Parallel-Trial Bit-Sieve Engine”. In: *Proceedings of the 11th International Conference on Complex, Intelligent, and Software Intensive Systems.* Vol. 611. Advances in Intelligent Systems and Computing. Springer, 2017, pp. 432–438. DOI: 10.1007/978-3-319-61566-0_39.

[41] Sascha Mücke, Nico Piatkowski, and Katharina Morik. “Hardware Acceleration of Machine Learning Beyond Linear Algebra”. In: *Proceedings of Machine Learning and Knowledge Discovery in Databases.* Vol. 1167. Communications in Computer and Information Science. Springer, 2019, pp. 342–347. DOI: 10.1007/978-3-030-43823-4_29.

[42] Abraham P. Punnen et al. *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications.* Springer Cham, 2022. ISBN: 978-3-031-04519-6. DOI: `10.1007/978-3-031-04520-2`.

[43] Panos M. Pardalos and Somesh Jha. "Complexity of Uniqueness and Local Search in Quadratic 0–1 Programming". In: *Operations research letters* 11.2 (1992), pp. 119–123.

[44] Eranda Çela and Abraham P. Punnen. "Complexity and Polynomially Solvable Special Cases of QUBO". In: *The Quadratic Unconstrained Binary Optimization Problem: Theory, Algorithms, and Applications.* Springer International Publishing, 2022, pp. 57–95. ISBN: 978-3-031-04520-2. DOI: `10.1007/978-3-031-04520-2_3`.

[45] Patrenahalli M. Narendra and Keinosuke Fukunaga. "A Branch and Bound Algorithm for Feature Subset Selection". In: *IEEE Trans. Computers* 26.9 (1977), pp. 917–922. DOI: `10.1109/TC.1977.1674939`.

[46] Daniel Rehfeldt, Thorsten Koch, and Yuji Shinano. "Faster exact solution of sparse MaxCut and QUBO problems". In: *Math. Program. Comput.* 15.3 (2023), pp. 445–470. DOI: `10.1007/S12532-023-00236-6`.

[47] Scott Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. "Optimization by Simulated Annealing". In: *Science* 220.4598 (1983), pp. 671–680. DOI: `10.1126/science.220.4598.671`.

[48] Fred Glover and Manuel Laguna. "Tabu search". In: *Handbook of combinatorial optimization.* Springer, 1998, pp. 2093–2229.

[49] David E. Goldberg and Chie Hsiung Kuo. "Genetic algorithms in pipeline optimization". In: *Journal of Computing in Civil Engineering* 1.2 (1987), pp. 128–141.

[50] Gary A. Kochenberger et al. "The unconstrained binary quadratic programming problem: a survey". In: *J. Comb. Optim.* 28.1 (2014), pp. 58–81. DOI: `10.1007/S10878-014-9734-0`.

[51] Stephen G. Brush. "History of the Lenz-Ising Model". In: *Rev. Mod. Phys.* 39 (4 1967), pp. 883–893. DOI: `10.1103/RevModPhys.39.883`.

[52] Ernst Ising. "Beitrag Zur Theorie Des Ferromagnetismus". In: *Zeitschrift für Physik* 31 (1925), pp. 253–258.

[53] Sotiris B. Kotsiantis, Dimitris Kanellopoulos, and Panagiotis E. Pintelas. "Data preprocessing for supervised leaning". In: *International journal of computer science* 1.2 (2006), pp. 111–117.

[54] Laurens Van Der Maaten, Eric O. Postma, H. Jaap Van Den Herik, et al. "Dimensionality reduction: A comparative review". In: *Journal of Machine Learning Research* 10.66-71 (2009), p. 13.

[55] Girish Chandrashekar and Ferat Sahin. "A survey on feature selection methods". In: *Comput. Electr. Eng.* 40.1 (2014), pp. 16–28. DOI: `10.1016/J.COMPELECENG.2013.11.024`.

[56] George H. John, Ron Kohavi, and Karl Pfleger. "Irrelevant Features and the Subset Selection Problem". In: *Proceedings of the Eleventh International Conference.* Morgan Kaufmann, 1994, pp. 121–129. DOI: `10.1016/B978-1-55860-335-6.50023-4`.

[57] R. Leardi, R. Boggia, and M. Terrile. "Genetic algorithms as a strategy for feature selection". In: *Journal of Chemometrics* 6.5 (1992), pp. 267–281. DOI: 10.1002/cem.1180060506.

[58] W. Siedlecki and J. Sklansky. "A note on genetic algorithms for large-scale feature selection". In: *Handbook of Pattern Recognition and Computer Vision*. World Scientific, 1993, pp. 88–107. ISBN: 978-981-02-1136-3. DOI: 10.1142/9789814343138_0005.

[59] Corinna Cortes and Vladimir Vapnik. "Support-Vector Networks". In: *Machine Learning* 20.3 (1995), pp. 273–297. DOI: 10.1007/BF00994018.

[60] Vladimir Vapnik. *The Nature of Statistical Learning Theory*. Statistics for Engineering and Information Science. Springer, 2000. DOI: 10.1007/978-1-4757-3264-1.

[61] Maria Schuld and Francesco Petruccione. *Supervised learning with quantum computers*. Vol. 17. Springer, 2018. DOI: 10.1007/978-3-319-96424-9.

[62] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information (10th Anniversary edition)*. Cambridge University Press, 2016. ISBN: 978-1-10-700217-3.

[63] M. Cerezo et al. "Variational quantum algorithms". In: *Nature Reviews Physics* 3.9 (2021), pp. 625–644. DOI: 10.1038/s42254-021-00348-9.

[64] Yudong Cao et al. "Quantum chemistry in the age of quantum computing". In: *Chemical reviews* 119.19 (2019), pp. 10856–10915.

[65] Alberto Peruzzo et al. "A variational eigenvalue solver on a photonic quantum processor". In: *Nature Communications* 5.1 (2014), p. 4213. DOI: 10.1038/ncomms5213.

[66] M.-H. Yung et al. "From transistor to trapped-ion computers for quantum chemistry". In: *Scientific Reports* 4.1 (2014). DOI: 10.1038/srep03589.

[67] Jarrod R. McClean et al. "The theory of variational hybrid quantum-classical algorithms". In: *New Journal of Physics* 18.2 (2016), p. 023023. DOI: 10.1088/1367-2630/18/2/023023.

[68] Satoshi Morita and Hidetoshi Nishimori. "Mathematical foundation of quantum annealing". In: *Journal of Mathematical Physics* 49.12 (2008), p. 125210.

[69] Edward Farhi et al. *Quantum Computation by Adiabatic Evolution*. 2000. arXiv: quant-ph/0001106 [quant-ph].

[70] Christian Gruber. "Thermodynamics of systems with internal adiabatic constraints: time evolution of the adiabatic piston". In: *European journal of physics* 20.4 (1999), p. 259.

[71] Boris Altshuler, Hari Krovi, and Jeremie Roland. "Adiabatic quantum optimization fails for random instances of NP-complete problems". In: (2009). arXiv: 0908.2782 [quant-ph].

[72] Thore Gerlach and Sascha Mücke. "Investigating the Relation Between Problem Hardness and QUBO Properties". In: *Advances in Intelligent Data Analysis 22*. Vol. 14642. Lecture Notes in Computer Science. Springer, 2024, pp. 171–182. DOI: 10.1007/978-3-031-58553-1_14.

[73] Toby S. Cubitt, David Perez-Garcia, and Michael M. Wolf. "Undecidability of the spectral gap". In: *Nature* 528.7581 (2015), pp. 207–211.

[74] Amedeo Bertuzzi et al. “Evaluation of Quantum and Hybrid Solvers for Combinatorial Optimization”. In: *Proceedings of the 21st ACM International Conference on Computing Frontiers.* ACM, 2024. DOI: 10.1145/3649153.3649205.

[75] D-Wave Systems. *Error Sources for Problem Representation.* 2024. URL: https://docs.dwavesys.com/docs/latest/c_qpu_ice.html (visited on 12/12/2024).

[76] A. Robert Calderbank et al. “Quantum Error Correction Via Codes Over GF(4)”. In: *IEEE Trans. Inf. Theory* 44.4 (1998), pp. 1369–1387. DOI: 10.1109/18.681315.

[77] Frank Arute et al. “Quantum supremacy using a programmable superconducting processor”. In: *Nature* 574.7779 (2019), pp. 505–510. DOI: 10.1038/s41586-019-1666-5.

[78] Kristan Temme, Sergey Bravyi, and Jay M. Gambetta. “Error mitigation for short-depth quantum circuits”. In: *Physical review letters* 119.18 (2017).

[79] Ali Javadi-Abhari et al. *Quantum computing with Qiskit.* 2024. DOI: 10.48550/arXiv.2405.08810. arXiv: 2405.08810 [quant-ph].

[80] Sengthai Heng et al. “Decomposition Analysis of Quantum Native Gates on Various Quantum Computers”. In: *Proceedings of the 37th International Technical Conference on Circuits/Systems, Computers and Communications.* 2022, pp. 1–3. DOI: 10.1109/ITC-CSCC55581.2022.9894863.

[81] Google Quantum AI and Collaborators. “Quantum error correction below the surface code threshold”. In: *Nature* (2024). DOI: 10.1038/s41586-024-08449-y.

[82] Benedikt Fauseweh. “Quantum many-body simulations on digital quantum computers: State-of-the-art and future challenges”. In: *Nature Communications* 15.1 (2024). DOI: 10.1038/s41467-024-46402-9.

[83] Christa Zoufal, Aurélien Lucchi, and Stefan Woerner. “Variational quantum Boltzmann machines”. In: *Quantum Mach. Intell.* 3.1 (2021), pp. 1–15. DOI: 10.1007/S42484-020-00033-7.

[84] Maiyuren Srikumar, Charles D. Hill, and Lloyd C. L. Hollenberg. “Clustering and enhanced classification using a hybrid quantum autoencoder”. In: *Quantum Science and Technology* 7.1 (2021). DOI: 10.1088/2058-9565/ac3c53.

[85] Bu-Qing Chen and Xu-Feng Niu. “Quantum Neural Network with Improved Quantum Learning Algorithm”. In: *International Journal of Theoretical Physics* 59.7 (2020), pp. 1978–1991. DOI: 10.1007/s10773-020-04470-9.

[86] Juhyeon Kim, Joonsuk Huh, and Daniel K. Park. “Classical-to-quantum convolutional neural network transfer learning”. In: *Neurocomputing* 555 (2023). DOI: 10.1016/j.neucom.2023.126643.

[87] YaoChong Li et al. “A quantum deep convolutional neural network for image recognition”. In: *Quantum Science and Technology* 5.4 (2020). DOI: 10.1088/2058-9565/ab9f93.

[88] Andrea Ceschini, Antonello Rosato, and Massimo Panella. “Design of an LSTM Cell on a Quantum Hardware”. In: *IEEE Trans. Circuits Syst. II Express Briefs* 69.3 (2022), pp. 1822–1826. DOI: 10.1109/TCSII.2021.3126204.

[89] Joseph Bowles, Shahnawaz Ahmed, and Maria Schuld. *Better than classical? The subtle art of benchmarking quantum machine learning models.* 2024. arXiv: 2403.07059 [quant-ph].

[90] Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. “Estimating mutual information”. In: *Physical Review E* 69.6 (2004).

[91] Panagiotis Mandros et al. “Discovering Functional Dependencies from Mixed-Type Data”. In: *Proceedings of the 26th Conference on Knowledge Discovery and Data Mining*. ACM, 2020, pp. 1404–1414. DOI: 10.1145/3394486.3403193.

[92] Fred W. Glover, Mark W. Lewis, and Gary A. Kochenberger. “Logical and inequality implications for reducing the size and difficulty of quadratic unconstrained binary optimization problems”. In: *Eur. J. Oper. Res.* 265.3 (2018), pp. 829–842. DOI: 10.1016/J.EJOR.2017.08.025.

[93] Irene Rodríguez-Luján et al. “Quadratic Programming Feature Selection”. In: *J. Mach. Learn. Res.* 11 (2010), pp. 1491–1516. DOI: 10.5555/1756006.1859900.

[94] Soronzonbold Otgonbaatar and Mihai Datcu. “A Quantum Annealer for Subset Feature Selection and the Classification of Hyperspectral Images”. In: *IEEE J. Sel. Top. Appl. Earth Obs. Remote. Sens.* 14 (2021), pp. 7057–7065. DOI: 10.1109/JSTARS.2021.3095377.

[95] Zhimin He et al. “Quantum-enhanced feature selection with forward selection and backward elimination”. In: *Quantum Inf. Process.* 17.7 (2018), p. 154. DOI: 10.1007/S11128-018-1924-8.

[96] Yann LeCun and Corinna Cortes. *MNIST handwritten digit database*. 2010. URL: http://yann.lecun.com/exdb/mnist/ (visited on 12/12/2024).

[97] Vincent G. Sigillito et al. “Classification of radar returns from the ionosphere using neural networks”. In: *Johns Hopkins APL Technical Digest* 10.3 (1989), pp. 262–266.

[98] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml (visited on 12/12/2024).

[99] Leo Breiman et al. *Classification and Regression Trees*. Wadsworth, 1984. ISBN: 0-534-98053-8.

[100] Daniel Lewandowski, Dorota Kurowicka, and Harry Joe. “Generating random correlation matrices based on vines and extended onion method”. In: *J. Multivar. Anal.* 100.9 (2009), pp. 1989–2001. DOI: 10.1016/J.JMVA.2009.04.008.

[101] Isabelle Guyon. *Madelon*. UCI Machine Learning Repository. 2004. DOI: 10.24432/C5602H.

[102] M. Booth, S. P. Reinhardt, and A. Roy. *Partitioning optimization problems for hybrid classical/quantum execution*. Dwave Systems, 2017.

[103] F. Pedregosa et al. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[104] Isabelle Guyon et al. “Gene Selection for Cancer Classification using Support Vector Machines”. In: *Mach. Learn.* 46.1-3 (2002), pp. 389–422. DOI: 10.1023/A:1012487302797.

[105] Diederik P. Kingma and Max Welling. “Auto-Encoding Variational Bayes”. In: *Proceedings of the 2nd International Conference on Learning Representations*. 2014.

[106] Jürgen Schmidhuber. “Deep learning in neural networks: An overview”. In: *Neural Networks* 61 (2015), pp. 85–117.

[107] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *Proceedings of the 3rd International Conference on Learning Representations.* 2015.

[108] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, pp. 8024–8035.

[109] Leslie N. Smith and Nicholay Topin. *Super-Convergence: Very Fast Training of Neural Networks Using Large Learning Rates.* 2018. arXiv: `1708.07120 [cs.LG]`.

[110] James C. Spall. "An overview of the simultaneous perturbation method for efficient optimization". In: *Johns Hopkins apl technical digest* 19.4 (1998), pp. 482–492.

[111] Yoshifumi Nakata et al. "Unitary 2-designs from random X- and Z-diagonal unitaries". In: *Journal of Mathematical Physics* 58.5 (2017). DOI: `10.1063/1.4983266`.

[112] Dennis Willsch et al. "Support vector machines on the D-Wave quantum annealer". In: *Comput. Phys. Commun.* 248 (2020). DOI: `10.1016/J.CPC.2019.107006`.

[113] Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines". In: *ACM Trans. Intell. Syst. Technol.* 2.3 (2011), 27:1–27:27. DOI: `10.1145/1961189.1961199`.

[114] Ronald A. Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of eugenics* 7.2 (1936), pp. 179–188.

[115] R. Paul Gorman and Terrence J. Sejnowski. "Analysis of hidden units in a layered network trained to classify sonar targets". In: *Neural Networks* 1.1 (1988), pp. 75–89. DOI: `10.1016/0893-6080(88)90023-8`.

[116] Gintaras Palubeckis. "Multistart Tabu Search Strategies for the Unconstrained Binary Quadratic Optimization Problem". In: *Ann. Oper. Res.* 131.1-4 (2004), pp. 259–282. DOI: `10.1023/B:ANOR.0000039522.58036.68`.

[117] J. C. Platt. *Sequential minimal optimization: A fast algorithm for training support vector machines.* Tech. rep. Microsoft Research Technical Report, 1998.

[118] Tad Hogg et al. "Tools for Quantum Algorithms". In: *International Journal of Modern Physics C* 10.07 (1999), pp. 1347–1361. DOI: `10.1142/s0129183199001108`.

[119] Vivek V. Shende, Stephen S. Bullock, and Igor L. Markov. "Synthesis of quantum-logic circuits". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 25.6 (2006), pp. 1000–1010. DOI: `10.1109/TCAD.2005.855930`.

[120] R. Somma et al. "Simulating physical phenomena by quantum networks". In: *Phys. Rev. A* 65.4 (2002).

[121] Wassily Hoeffding. "Probability Inequalities for Sums of Bounded Random Variables". In: *Journal of the American Statistical Association* 58.301 (1963), pp. 13–30. DOI: `10.2307/2282952`.

[122] Paul D. Nation et al. "Scalable Mitigation of Measurement Errors on Quantum Computers". In: *PRX Quantum* 2 (4 2021). DOI: `10.1103/PRXQuantum.2.040326`.

[123] Sergey Bravyi et al. "Mitigating measurement errors in multiqubit experiments". In: *Physical Review A* 103.4 (2021). DOI: `10.1103/physreva.103.042605`.

[124] Rajeev Acharya et al. "Suppressing quantum errors by scaling a surface code logical qubit". In: *Nature* 614.7949 (2023), pp. 676–681. DOI: `10.1038/s41586-022-05434-1`.

[125] Leonardo Banchi and Gavin E. Crooks. "Measuring Analytic Gradients of General Quantum Evolution with the Stochastic Parameter Shift Rule". In: *Quantum* 5 (2021), p. 386. DOI: 10.22331/Q-2021-01-25-386.

[126] Yunchao Liu, Srinivasan Arunachalam, and Kristan Temme. "A rigorous and robust quantum speed-up in supervised machine learning". In: *Nature Physics* 17.9 (2021), pp. 1013–1017. DOI: 10.1038/s41567-021-01287-z.

[127] Jennifer R. Glick et al. "Covariant quantum kernels for data with group structure". In: *Nature Physics* 20.3 (2024), pp. 479–483. DOI: 10.1038/s41567-023-02340-9.

[128] Frank Arute et al. "Hartree-Fock on a superconducting qubit quantum computer". In: *Science* 369.6507 (2020), pp. 1084–1089. DOI: 10.1126/science.abb9811.

[129] Sam McArdle et al. "Variational ansatz-based quantum simulation of imaginary time evolution". In: *npj Quantum Information* 5.1 (2019), pp. 1–6.

[130] Edward Farhi and Hartmut Neven. *Classification with Quantum Neural Networks on Near Term Processors.* 2018. arXiv: 1802.06002 [quant-ph].

[131] Abhinav Kandala et al. "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets". In: *Nature* 549.7671 (2017), pp. 242–246.

[132] Ho Lun Tang et al. "Qubit-ADAPT-VQE: An Adaptive Algorithm for Constructing Hardware-Efficient Ansätze on a Quantum Processor". In: *PRX Quantum* 2.2 (2021). DOI: 10.1103/prxquantum.2.020310.

[133] Harper R. Grimsley et al. "An adaptive variational algorithm for exact molecular simulations on a quantum computer". In: *Nat. Commun.* 10.1 (2019), pp. 1–9.

[134] Gexiang Zhang. "Quantum-inspired evolutionary algorithms: a survey and empirical study". In: *J. Heuristics* 17.3 (2011), pp. 303–351. DOI: 10.1007/S10732-010-9136-0.

[135] Daniela Szwarcman, Daniel Civitarese, and Marley M. B. R. Vellasco. "Quantum-Inspired Neural Architecture Search". In: *Proceedings of the International Joint Conference on Neural Networks.* IEEE, 2019, pp. 1–8. DOI: 10.1109/IJCNN.2019.8852453.

[136] Lee Spector et al. "Genetic programming for quantum computers". In: *Genetic Programming* (1998), pp. 365–373.

[137] Georgiy Krylov and Martin Lukac. "Quantum encoded quantum evolutionary algorithm for the design of quantum circuits". In: *Proceedings of the 16th ACM International Conference on Computing Frontiers.* ACM, 2019, pp. 220–225. DOI: 10.1145/3310273.3322826.

[138] Donald A. Sofge. "Toward a framework for quantum evolutionary computation". In: *Proceedings of the IEEE Conference on Cybernetics and Intelligent Systems.* 2006, pp. 1–6.

[139] Rafael Lahoz-Beltra. "Quantum Genetic Algorithms for Computer Scientists". In: *Comput.* 5.4 (2016), p. 24. DOI: 10.3390/COMPUTERS5040024.

[140] Abhinav Anand, Matthias Degroote, and Alán Aspuru-Guzik. "Natural evolutionary strategies for variational quantum computation". In: *Machine Learning: Science and Technology* 2.4 (2021), p. 045012. DOI: 10.1088/2632-2153/abf3ac.

[141] Andrew Arrasmith et al. "Effect of barren plateaus on gradient-free optimization". In: *Quantum* 5 (2021), p. 558. DOI: 10.22331/q-2021-10-05-558.

[142] Ernesto Campos, Aly Nasrallah, and Jacob Biamonte. “Abrupt transitions in variational quantum circuit training”. In: *Physical Review A* 103.3 (2021). DOI: 10.1103/physreva.103.032607.

[143] Kenneth O. Stanley and Risto Miikkulainen. “Evolving Neural Networks through Augmenting Topologies”. In: *Evolutionary Computation* 10.2 (2002), pp. 99–127. DOI: 10.1162/106365602320169811.

[144] Adriano Barenco et al. “Elementary gates for quantum computation”. In: *Physical Review A* 52.5 (1995), pp. 34–57.

[145] Günter Rudolph. “Convergence of Evolutionary Algorithms in General Search Spaces”. In: *Proceedings of the IEEE International Conference on Evolutionary Computation.* IEEE, 1996, pp. 50–54. DOI: 10.1109/ICEC.1996.542332.

[146] Pierre Pfeuty. “The one-dimensional Ising model with a transverse field”. In: *Annals of Physics* 57.1 (1970), pp. 79–90.

[147] David Wierichs, Christian Gogolin, and Michael Kastoryano. “Avoiding local minima in variational quantum eigensolvers with the natural gradient optimizer”. In: *Physical Review Research* 2.4 (2020). DOI: 10.1103/physrevresearch.2.043246.

[148] David Sherrington and Scott Kirkpatrick. “Solvable model of a spin-glass”. In: *Physical Review letters* 35.26 (1975).

[149] Kevin J. Sung et al. “Using models to improve optimizers for variational quantum algorithms”. In: *Quantum Science and Technology* 5.4 (2020).

[150] Alessandro Giovagnoli et al. “QNEAT: Natural Evolution of Variational Quantum Circuit Architecture”. In: *Companion Proceedings of the Conference on Genetic and Evolutionary Computation.* ACM, 2023, pp. 647–650. DOI: 10.1145/3583133.3590675.

[151] Giovanni Acampora et al. “EVOVAQ: EVOlutionary algorithms-based toolbox for VAriational Quantum circuits”. In: *SoftwareX* 26 (2024), p. 101756. DOI: 10.1016/J.SOFTX.2024.101756.

[152] Leo Sünkel et al. *GA4QCO: Genetic Algorithm for Quantum Circuit Optimization.* 2023. arXiv: 2302.01303 [quant-ph].

[153] Troels F. Rønnow et al. “Defining and Detecting Quantum Speedup”. In: *Science* 345.6195 (2014), pp. 420–424. DOI: 10.1126/science.1252319.

[154] *Information technology – Microprocessor Systems – Floating-Point arithmetic.* Standard. International Organization for Standardization, 2020.

[155] D-Wave Systems. *Technical Description of the D-Wave Quantum Processing Unit.* User manual 09-1109A-Y. 2021, p. 62.

[156] G. Ballou. *Handbook for Sound Engineers.* Focal Press. Focal, 2005. DOI: 10.4324/9780080927619.

[157] Endre Boros, Peter L. Hammer, and Gabriel Tavares. *Preprocessing of unconstrained quadratic binary optimization.* Tech. rep. Rutgers University, 2006.

[158] Mark W. Lewis and Fred W. Glover. “Quadratic unconstrained binary optimization problem preprocessing: Theory and empirical analysis”. In: *Networks* 70.2 (2017), pp. 79–97. DOI: 10.1002/NET.21751.

[159] Endre Boros et al. “A max-flow approach to improved lower bounds for quadratic unconstrained binary optimization (QUBO)”. In: *Discret. Optim.* 5.2 (2008), pp. 501–529. DOI: 10.1016/J.DISOPT.2007.02.001.

[160] Peter L. Hammer, Pierre Hansen, and Bruno Simeone. “Roof duality, complementation and persistency in quadratic 0-1 optimization”. In: *Math. Program.* 28.2 (1984), pp. 121–155. DOI: 10.1007/BF02612354.

[161] David Biesner et al. “Solving Subset Sum Problems using Quantum Inspired Optimization Algorithms with Applications in Auditing and Financial Data Analysis”. In: *Proceedings of the 21st IEEE International Conference on Machine Learning and Applications.* IEEE, 2022, pp. 903–908. DOI: 10.1109/ICMLA55696.2022.00150.

[162] Norman L. Johnson, Samuel Kotz, and Narayanaswamy Balakrishnan. *Continuous univariate distributions, volume 2.* Vol. 289. John wiley & sons, 1995. ISBN: 978-0-471-58494-0.

[163] Richard Kaye. “Minesweeper Is NP-complete”. In: *Mathematical Intelligencer* 22.2 (2000), pp. 9–15.

[164] Joseph Culberson. *Sokoban is PSPACE-complete.* Tech. rep. Department of Computing Science, University of Alberta, 1997.

[165] Steve Nebel, Sascha Schneider, and Günter Daniel Rey. “Mining Learning and Crafting Scientific Experiments: A Literature Review on the Use of Minecraft in Education and Research”. In: *J. Educ. Technol. Soc.* 19.2 (2016), pp. 355–366.

[166] Kouki Yonaga, Masamichi J. Miyama, and Masayuki Ohzeki. *Solving Inequality-Constrained Binary Optimization Problems on Quantum Annealer.* 2020. arXiv: 2012.06119 [quant-ph].

[167] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms.* 5th. Springer Publishing Company, Incorporated, 2012. ISBN: 3642244874.

[168] Tomás Vyskocil, Scott Pakin, and Hristo N. Djidjev. “Embedding Inequality Constraints for Quantum Annealing Optimization”. In: *Quantum Technology and Optimization Problems.* Vol. 11413. Lecture Notes in Computer Science. Springer, 2017, pp. 11–22. DOI: 10.1007/978-3-030-14082-3_2.

[169] Stephen P. Boyd et al. “Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers”. In: *Found. Trends Mach. Learn.* 3.1 (2011), pp. 1–122. DOI: 10.1561/2200000016.

[170] Michael J. D. Powell. “A method for nonlinear constraints in minimization problems”. In: *Optimization* (1969), pp. 283–298.

[171] Daniel Gabay and Bertrand Mercier. “A dual algorithm for the solution of nonlinear variational problems via finite element approximation”. In: *Computers & mathematics with applications* 2.1 (1976), pp. 17–40.

[172] Ken Perlin. “An image synthesizer”. In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques.* ACM, 1985, pp. 287–296. DOI: 10.1145/325334.325247.

[173] Andrew Lucas. “Ising formulations of many NP problems”. In: *Frontiers in Physics* 2 (2014). DOI: 10.3389/fphy.2014.00005.

[174] Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. “A Survey of NP-Complete Puzzles”. In: *J. Int. Comput. Games Assoc.* 31.1 (2008), pp. 13–34.

[175] Luciano Gualà, Stefano Leucci, and Emanuele Natale. “Bejeweled, Candy Crush and Other Match-Three Games Are (NP-)Hard”. In: *Proceedings of the IEEE Conference on Computational Intelligence and Games*. 2014, pp. 1–8. DOI: 10.1109/CIG.2014.6932866.

[176] Takayuki Yato and Takahiro Seta. “Complexity and Completeness of Finding Another Solution and Its Application to Puzzles”. In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 86-A.5 (2003), pp. 1052–1060.

[177] Christian Bauckhage, Fabrice Beaumont, and Sebastian Müller. “ML2R Coding Nuggets Hopfield Nets for Sudoku”. In: (2021).

[178] Timothy Resnick. *Sudoku at the intersection of classical and quantum computing*. Tech. rep. Department of Computer Science, The University of Auckland, New Zealand, 2014.

[179] Peter J. M. van Laarhoven and Emile H. L. Aarts. *Simulated Annealing: Theory and Applications*. Vol. 37. Mathematics and Its Applications. Springer, 1987. DOI: 10.1007/978-94-015-7744-1.

[180] Thomas Pochart, Paulin Jacquot, and Joseph Mikael. “On the challenges of using D-Wave computers to sample Boltzmann Random Variables”. In: *Companion Proceedings of the IEEE 19th International Conference on Software Architecture*. IEEE, 2022, pp. 137–140. DOI: 10.1109/ICSA-C54293.2022.00034.

[181] Gary McGuire, Bastian Tugemann, and Gilles Civario. “There Is No 16-Clue Sudoku: Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration”. In: *Exp. Math.* 23.2 (2014), pp. 190–217. DOI: 10.1080/10586458.2013.870056.

[182] P. Wiederkehr and T. Siebrecht. “Virtual Machining: Capabilities and Challenges of Process Simulations in the Aerospace Industry”. In: *Procedia Manufacturing* 6 (2016), pp. 80–87. DOI: 10.1016/j.promfg.2016.11.011.

[183] Berend Denkena, Thilo Grove, and Alexander Krödel. “A New Tool Concept for Milling Automotive Components”. In: *Procedia CIRP* 46 (2016), pp. 444–447. DOI: 10.1016/j.procir.2016.04.055.

[184] António Festas, António Ramos, and Jo
textasciitilde ao Paulo Davim. “Machining of titanium alloys for medical application-a review”. In: *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture* 236.4 (2022), pp. 309–318.

[185] Maria Schuld, Ryan Sweke, and Johannes Jakob Meyer. “Effect of data encoding on the expressive power of variational quantum-machine-learning models”. In: *Physical Review A* 103.3 (2021). DOI: 10.1103/physreva.103.032430.

[186] Kun Liu et al. “Precision manufacturing of key components for an ultra miniature gas turbine unit for power generation”. In: *Microsystem Technologies* 15 (2009), pp. 1417–1425.

[187] Morad Mohammed Abdelhalim, Benhabib Mohamed Choukri, and Meliani Sidi Mohammed. “Optimization and Design of Low-Power Wind Turbine Blades”. In: *Proceedings of the 2nd International Conference on Electrical Engineering and Automatic Control*. 2024, pp. 1–6. DOI: 10.1109/ICEEAC61226.2024.10576513.

[188] Mikell P. Groover. *Fundamentals of modern manufacturing: materials, processes, and systems.* John Wiley & Sons, 2010. ISBN: 978-0470467008.

[189] Yusuf Altintas. *Manufacturing automation: metal cutting mechanics, machine tool vibrations, and CNC design.* Cambridge University Press, 2012. ISBN: 978-0-521-17247-9.

[190] Yusuf Altintas et al. "Chatter stability of machining operations". In: *Journal of Manufacturing Science and Engineering* 142.11 (2020).

[191] Amal Saadallah et al. "Stability prediction in milling processes using a simulation-based Machine Learning approach". In: *Procedia CIRP* 72 (2018), pp. 1493–1498. DOI: 10.1016/j.procir.2018.03.062.

[192] Felix Finkeldey et al. "Real-time prediction of process forces in milling operations using synchronized data fusion of simulation and sensor data". In: *Eng. Appl. Artif. Intell.* 94 (2020). DOI: 10.1016/J.ENGAPPAI.2020.103753.

[193] Petra Wiederkehr, Felix Finkeldey, and Tobias Siebrecht. "Reduction of experimental efforts for predicting milling stability affected by concept drift using transfer learning on multiple machine tools". In: *CIRP Annals* 73.1 (2024), pp. 301–304. DOI: 10.1016/j.cirp.2024.04.084.

[194] Dong C. Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Math. Program.* 45.1-3 (1989), pp. 503–528. DOI: 10.1007/BF01589116.