UNIVERSITÀ
DI TRENTO
Department of
Information Engineering and Computer Science

Doctoral School in
Information and Communication Technology

Optimizing Quantum Circuit Layout using Quantum Annealing

Mariia Maltseva

Advisor
Assoc. Prof. Enrico Blanzieri
University of Trento

Academic year 2023/2024

Acknowledgements

First of all, I would like to appreciate my supervisor, Professor Enrico Blanzieri, for guiding me in my study of quantum computing and for supporting my research endeavors. I would also like to thank Professor Alexander Rumyantsev for his valuable recommendations during the research process, as well as the reviewers of this PhD thesis for their constructive feedback.
Additionally, I would like to extend my heartfelt thanks to my family – my mother Oksana, my father Alexei, and my husband Alexei – for their warm support throughout my research journey.

Abstract

Quantum computing has garnered significant interest due to its theoretical potential for exponential speedup in computations. This advantage arises from the unique properties of quantum mechanics, which enable quantum bits (qubits) to exist in multiple states simultaneously and to become entangled. However, the current Noisy Intermediate-Scale Quantum Computing era is constrained by the limited number of available qubits and their high sensitivity to environmental noise.

In this thesis, we consider a quantum circuit-based model, recognized for its universality in encoding any computation. Nevertheless, leveraging this model requires adherence to the constraints imposed by quantum hardware. One critical limitation is the Nearest Neighbor condition, prevalent in quantum hardware architectures, which necessitates that computations should be performed solely on adjacent qubits. Consequently, additional swapping operations, or SWAP gates, are required to make interacting qubits neighboring. Given the high computational cost associated with these SWAP gates, it is essential to minimize their usage. To address this challenge, we introduce two hybrid quantum-quantum approaches designed to adapt quantum circuits for the Noisy Intermediate-Scale Quantum era by utilizing quantum computing technologies. Both approaches are based on the qubit line reordering technique and use Quantum Annealing in the optimization part. The first approach optimizes a quantum circuit by global qubit line reordering based on the Graph Partitioning problem formulation. The second approach implements local qubit line reordering using Boolean satisfiability theory. As our problem-solving tool, we use Quantum Annealing, leveraging its capabilities as a quantum heuristic optimization solver. Our numerical experiments, conducted with several benchmark quantum circuits, demonstrate promising results in optimizing the circuits.

Keywords

[Quantum circuit optimization, Quantum Annealing, Nearest Neighbor, Graph Partitioning, Boolean Satisfiability]

Contents

List of Tables

List of Figures

Glossary

AQC Adiabatic quantum computing.

GF Galois field.

GP Graph partitioning.

LNN Linear Nearest Neighbor.

MoS Metric of success.

NISQ Noisy Intermediate-Scale Quantum.

NN Nearest Neighbor.

NNC Nearest Neighbor Cost.

QA Quantum Annealing.

QALS Quantum Annealing Learning Search.

QAOA Quantum Approximate Optimization Algorithm.

QC Quantum circuit.

QUBO Quadratic Unconstrained Binary Optimization.

SA Simulated Annealing.

SAT Boolean satisfiability.

TEHQO Tabu-Enhanced Hybrid Quantum Optimization.

TEQAHQO Tabu-Enhanced Quantum Annealing Hybrid Quantum Optimization.

Chapter 1

Introduction

Nowadays, quantum computing becomes one of the innovative areas of research. It is grounded in the principles of quantum mechanics, which offer the potential for superior speed compared to conventional computing. The key difference from the classical computing is that traditional computers operate using bits, that can only be in one of two state (0 or 1) at any time, whereas quantum computers process quantum bits, or qubits, which can be in multiple states at once, thanks to superposition. This quantum property, combined with quantum entanglement, enables one to process multiple possibilities concurrently, making quantum computing incredibly powerful for certain types of problems. A remarkable example of this is the Shor's algorithm [105] to perform integer factoring, which completes the task in polynomial time while classical algorithms at best need superpolynomial time [26].

Quantum computing has the potential to transform multiple industries by tackling complex problems that are currently out of the reach of traditional computers. It is known that combinatorial optimization problems pose substantial difficulties in different fields, and traditional computing often struggles with their exponential complexity [108]. One of them is the job-shop scheduling problem. It has a significant practical impact on manufacturing [128] and becomes intractable because of the size of real-life problems [21]. However, Quantum Annealing (QA) algorithm is proposed to solve a problem with greater scalability and solution quality [21]. In addition, the NP-hard Travelling Salesman Problem is widely used in the manufacturing and routing industries [51], and several quantum approaches have been introduced to solve it [61, 51]. Quantum algorithms like the Quantum Approximate Optimization Algorithm (QAOA) can efficiently solve complex optimization problems, with applications in logistics [95], finance, and more. Quantum machine learning combines classical and quantum algorithms to

improve pattern recognition [28] and data analysis tasks. Quantum computers also have the ability to simulate other quantum systems with greater efficiency than classical computers [45], which is essential for understanding and developing new materials, chemicals, and technologies.

However, quantum computing is still in its early stages (named as Noisy Intermediate-Scale Quantum (NISQ) era [96]), and there are significant challenges to overcome in order to realize the full potential of quantum computing. First, due to the *high sensitivity of qubits to the surrounding environment*, quantum computations should be performed at temperature near absolute 0 [68] for the most mature quantum computing technologies based on superconducting and trapped ion qubits. Cold temperature allows qubits to stay coherent, i.e. to maintain superposition and entanglement [127] in order to perform complex quantum computations. However, it is hardly possible because of the interactions with the environment (e.g., during the qubit state measurement). Due to this environmental noise, temperature requirements are violated resulting in a loss of qubit coherence. In addition, it leads to loss of quantum properties [99] and occurrence of computational errors. The problem of unreliable quantum computing imposes constraints on the application of quantum computing and quantum error correction is actively progressing with various techniques [106, 67, 5] that have already been proposed to solve this problem. Furthermore, due to the limited coherence of qubits (lasting from milliseconds to seconds [20]), NISQ devices have the so-called *coupling constraint* [25] meaning that quantum operations can be performed only on qubits that are physically close to each other (neighbors). For that, Nearest Neighbor (NN) architecture is commonly used as the physical connectivity topology. In case when local qubits interacting within a quantum circuit (QC) are non-adjacent, they need to be physically remapped [25], and the mapping problem has already been extensively studied for linear architecture [97, 24, 101, 94] as well as for two-dimensional [114] and three-dimensional topologies [16]. Besides that, quantum hardware is constrained by a *limited number of qubits* (e.g. 1121 qubits in largest IBM's quantum processor Condor [92] and D-Wave Advantage 2 with 7000+ qubits [84]). Despite these hurdles, the field is progressing rapidly, driven by both academic research and private sector investment.

There are several models of quantum computing, among which the QC model is a commonly used one because of its universality [74], meaning that any unitary operation can be implemented as a QC [100]. Similar to classical circuits in traditional computing, QCs serve as the building blocks of quantum algorithms; they consist of quantum gates arranged sequentially to act on logical qubits. To be implemented on

quantum hardware, a QC (or unitary operation) should be decomposed into a sequence of elementary gates native to the corresponding hardware [130]. Moreover, as previously mentioned, quantum hardware architectures often impose physical constraints, e.g. NN condition that allows one to apply a gate only to qubits that are neighboring physically. Consequently, logical qubits within a QC must be mapped to neighboring physical qubits that are embedded in the hardware architecture. To facilitate this mapping, an auxiliary two-qubit $SWAP$ gate ($SWAP$) is used to swap the positions of a pair of interacting logical qubits, ensuring that they occupy neighboring physical qubit positions. Given the high cost associated with using $SWAP$ gates, it is reasonable to reduce its count, and numerous approaches have already been proposed to achieve it. These approaches vary based on the way of QC representation and optimization technique used. For instance, circuits are often studied as NN-compliant circuits with heuristics used to minimize $SWAP$ count (ant colony metaheuristic [16]). In [24], graph bipartitioning is applied to the qubit line adjacency graph to rearrange the qubit lines in a manner that reduces the $SWAP$ count required in a linear (one-dimensional) NN (LNN) architecture. In [7], QC is represented as a graph of interactions between each control-target pair as an edges with weights equal to the number of times this pair appears in the circuit, and $SWAP$ count is minimized by qubit line rearrangement in two-dimensional (2D) NN architecture using Harmony-Search algorithm being a sort of gradient-based metaheuristic.

Despite global qubit line rearrangement, qubit lines can be rearranged locally, implying that qubits used in the description of only the current gate (or set of gates) are reordered. In [17], the authors introduce look-ahead heuristics based on selecting the best permutation of qubits (depending on its overall cost) to make current gate NN-compliant. In [114], Boolean satisfiability (SAT) together with A^* search are used to reorder the qubits locally with minimal $SWAP$s needed. More detailed description of various QC optimization techniques will be presented in Section 2.3.

In the thesis, we concentrate on qubit reordering schemes and construct two hybrid quantum-quantum QC optimization techniques based on global and local qubit line reordering following approaches from [24] and [114], respectively. For that, we replace classical component of the algorithms with QA that is a quantum technique to solve optimization problems. The choice of QA as a quantum component of the approach is motivated by its ability to find global optimum due to the quantum mechanics properties, particularly the quantum tunneling effect which enables the algorithm to escape local optima. Additionally, QA has not been applied in this context yet. Moreover, QA has a higher stage in the development than a QC model (e. g. QA implemented

in `D-Wave Advantage_system5.4` is equipped with 5614 qubits and 40050 couplers whereas IMB's Condor has only 1121 qubits) but it is non-universal. However, as QA is a heuristic (because of environmental noise in quantum computing), an important practical issue is its convergence that has already been proven [85].

In Chapter 3, we propose a modification of classical approach to global qubit line reordering [24] which is based on Graph Partitioning (GP) of qubit line adjacency graph and aims to minimize $SWAP$ count in LNN architecture. Instead of using a classical bipartitioning tool, we propose using QA which allows us to extend the approach to k-partitioning. We also verify the feasibility of this approach by conducting numerical experiments on several QCs.

In Chapter 4, we focus on optimizing QCs for the 2D NN architecture following the approach proposed in [114]. In this work, the authors treat a circuit as a gate dependency graph and partition it into NN-compliant subcircuits through local reordering of qubits. They propose using Boolean satisfiability (which has already been studied in quantum computing [73, 18, 15, 88]) for partitioning the graph. To connect 2D placements of qubits obtained for each subcircuit, the A^* search is used. The novelty of our approach lies in proposing a hybrid quantum-quantum QC optimization strategy where QA replaces the classical counterpart.

Additionally, we study tabu-enhanced QA [90] that represents an iterative QA method that penalizes non-optimal solutions by introducing tabu mechanism. We consider the convergence issue and apply tabu-enhanced QA in the local qubit reordering approach (proposed in Chapter 4) instead of the original QA.

The structure of the thesis is organized as follows. In Chapter 2, we present the fundamental concepts of quantum computing that underpin our research together with the overview of various QC optimization techniques. Chapters 3 and 4 introduce and explore QC optimization techniques based on global and local qubit reordering, and we validate these approaches by conducting numerical experiments to assess their applicability. In Chapter 5, we investigate the tabu-enhanced QA approach, addressing convergence issue and applying the approach to optimize QCs. Finally, in the Conclusion, we summarize the key findings and results of the current research.

Chapter 2

Background

In this chapter, we present fundamental concepts of quantum computing, including the definitions of qubit, quantum superposition, and entanglement. We also explore QA selected as our problem-solver, detailing the format for problem instances that it can accept. Additionally, we discuss GP, which will be considered as an optimization tool in the subsequent sections of this thesis.

2.1 Fundamentals of Quantum Computing

2.1.1 Quantum Theory Properties

Quantum bit (qubit) is a basic information storage unit in quantum computing systems. It is known that classical bit has the two basic Boolean states (0 and 1) and can be at one of them at a time, whereas a qubit can store information as a *superposition*, i.e. linear combination of the two basic Boolean states $|0\rangle \equiv \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ and $|1\rangle \equiv \begin{bmatrix} 0 \\ 1 \end{bmatrix}$. Thus, the qubit state $|\phi\rangle$ is described in two-dimensional Hilbert space $\mathcal{H}$ as follows:

$$|\phi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix} = \alpha\,|0\rangle + \beta\,|1\rangle\,, \tag{2.1}$$

where $\alpha, \beta \in \mathbb{C}$ such that $|\alpha|^2 + |\beta|^2 = 1$. This phenomenon allows qubits to exist simultaneously in several states, implying a possible computational speedup [111]. However, to obtain a specific value of the qubit, a *measurement* is performed, which returns (and at the same time converts the qubit into) one of the basic states, i.e., $|0\rangle$ and $|1\rangle$ with probability $|\alpha|^2$ and $|\beta|^2$, respectively, for the qubit in state $|\phi\rangle$ given in (2.1).

To enhance understanding, a visual representation of a qubit is provided. The Bloch sphere [57] serves this purpose allowing a qubit to be represented as a point (see Figure 2.1) on its surface with

$$|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + \exp\{i\phi\}\sin\left(\frac{\theta}{2}\right)|1\rangle\,.$$

The top and bottom of the Bloch sphere correspond to $|0\rangle$ and $|1\rangle$, respectively. Other points marked in Figure 2.1 correspond to different possible superpositional states of a single qubit. For example, states $|+\rangle = \frac{|0\rangle+|1\rangle}{2}$ and $|-\rangle = \frac{|0\rangle-|1\rangle}{2}$, $|i\rangle = \frac{|0\rangle+i|1\rangle}{2}$ and $|-i\rangle = \frac{|0\rangle-i|1\rangle}{2}$ are different superpositions of $|0\rangle$ and $|1\rangle$.

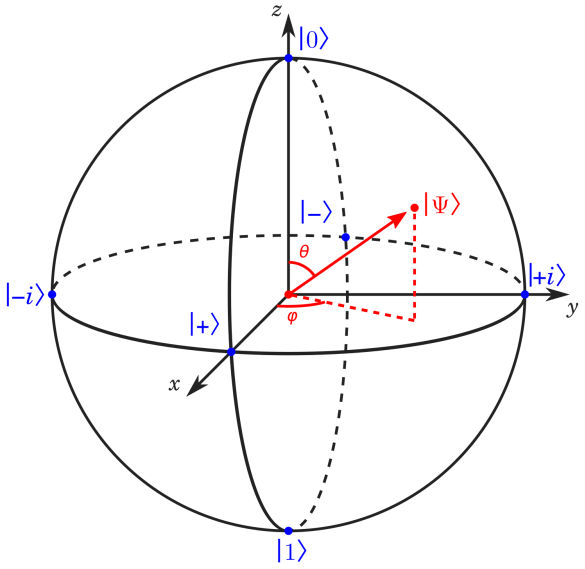


Figure 2.1: Bloch sphere.

Another quantum distinct feature is quantum *entanglement* [57]. This feature allows qubits to interact with one another as groups [52], thus connecting their states. Therefore, in general, a multi-qubit (e.g., n-qubit) system state is described as a linear combination

$$\sum_{i=0}^{2^n-1} \alpha_i\,|i\rangle$$

of the basic n-qubit states $|0,\ldots,0\rangle, \ldots, |1,\ldots,1\rangle$ which are sometimes given in their corresponding integer form as $|0\rangle, |1\rangle, \ldots, |2^n-1\rangle$, with complex coefficients $\alpha_0,\ldots,\alpha_{2^n-1} \in \mathbb{C}$ such that $|\alpha_0|^2 + \cdots + |\alpha_{2^n-1}|^2 = 1$. Such a combination can also be written in the form of a 2^n-component column vector $[\alpha_0,\ldots,\alpha_{2^n-1}]^T$. Measurement of the state of a qubit enables one to obtain information about a qubit that is entangled.

Note that when adding more qubits, quantum computers can process more information and solve more complex problems.

2.1.2 Quantum Gates

In the field of quantum computing, quantum gates are analogous to classical logic gates. Quantum computers use quantum gates to operate on qubits, whereby the states of the qubits are changed depending on the specific gate applied. Quantum gates are combined to form QCs, which finally implement the desired algorithms. Mathematically, a quantum gate can be represented as a unitary operator, meaning that its inverse and adjoint matrices are equal. Unitarity guarantees the conservation of the sum of probabilities for a quantum system to be in a specific state, making quantum computations reversible. For a single-qubit quantum system, the state of a qubit is described by a superposition (2.1). The unitary operators can only act on the qubit in a way that preserves the total probability of the qubit being in some state [57]. Thus, the unitarity condition provides us with the reversibility of quantum gates, because the action of a gate can be undone by applying its adjoint. This property is applied not only to single-qubit gates, but also to gates acting on many qubits as well. Note also that the application of the gate to the qubits can be considered as the (left) multiplication of the vector of the qubit state by the corresponding unitary matrix.

Theoretically, a QC can be viewed as a single quantum gate that operates on all qubits within a system. However, researchers are limited to a specific set of gates, such as single-qubit and two-qubit gates, on the practical side. Various gates are assembled into the so-called gate sets or gate libraries, which are basic pieces of a quantum computing program for various architectures and implementations of quantum computing principles. In the thesis, we consider the so-called NCT gate library, i.e. library consisting of *NOT*, *CNOT*, and *Toffoli* gates which are defined below. Although *Toffoli* gate is sufficient for synthesizing any reversible Boolean function, *NOT* together with *CNOT* provide implementional simplicity [13]. In the context of NISQ devices, *CNOT* is known to be susceptible to errors [60], prompting researchers to minimize its usage in circuit implementations. Additionally, due to a common NN connectivity constraint of NISQ devices, interacting qubits should be adjacent. This requirement often necessitates the use of auxiliary $SWAP$ gates which are typically implemented using three *CNOT*, further compounding the error susceptibility. As a result, *CNOT* count is a commonly used metric of QC optimization, and, in particular case of NN constraint, reducing the number of $SWAP$ gates is particularly important while optimizing QCs.

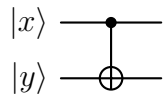
The single-qubit NOT gate is described by a matrix

$$\sigma_X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix},$$

and such a gate turns the qubit state $[\alpha, \beta]^T$ into the state $[\beta, \alpha]^T$, i.e. after applying such a gate, pure state $|0\rangle$ becomes $|1\rangle$ and vice versa. As such, the NOT gate inverts the qubit state.

The controlled-NOT gate, $CNOT$, is a two-qubit gate which inverts the so-called *target* (e.g. the second) qubit if the *control* (the first) qubit is in state $|1\rangle$. Such a $CNOT$ gate is defined by the unitary matrix

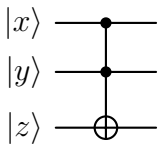
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

and represented as follows 

where $\oplus$ is the sum modulo 2.

Toffoli gate is an extension of the $CNOT$ gate having two control bits (say, the first two) and a target bit (say, the last), which is inverted once *both* control bits are in state $|1\rangle$. It is defined by the matrix

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

and represented as follows 

Note that the NCT gate set is complete for the so-called Boolean reversible circuits, i.e. the QCs that convert Boolean input into Boolean output [81].

Besides that, there is a fault-tolerant single-qubit gate set consisting of the Hadamard (H) and T gates. H gate is defined as

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

and performs π-rotation of a quantum state around Z axis transforming $|0\rangle$ into $H|0\rangle = |+\rangle$ and $H|1\rangle = |-\rangle$. T gate is defined as

$$T = \begin{bmatrix} 1 & 0 \\ 0 & \exp\{i\pi/4\} \end{bmatrix}$$

and applies phase shift to a qubit state if it is $|1\rangle$ and leaves $|0\rangle$ unchanged.

Note that any single-qubit gate along with $CNOT$ constructs a universal quantum gate library [87] which means that any computation can be performed using these gates.

2.2 Quantum Computing Models

2.2.1 Quantum Circuit Model

Like a logic circuit consists of logic gates, a QC is constructed by combining quantum gates in a sequence. It is a universal and popular quantum computing model as it allows to direct a quantum machine at any complex computation [115]. In general, QC model is defined as a sequence of quantum gates U_i applied to some initial state $|\psi\rangle$ returning $|\psi_f\rangle$. However, quantum machines have extremely restricted resources: modern IBM's quantum machine Condor has 1121 superconducting qubits [22] limiting the size of QC applied.

As a general property of quantum systems, composition of multiple qubits as well as multi-qubit circuits are performed via the Kronecker product of matrices:

$$\boldsymbol{A} \otimes \boldsymbol{B} = \begin{pmatrix} a_{11}\boldsymbol{B} & \dots & a_{1n}\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ a_{n1}\boldsymbol{B} & \dots & a_{nn}\boldsymbol{B} \end{pmatrix}.$$

In particular, considering the product $\psi \otimes \varphi$ of two single-qubit states $|\psi\rangle$ and $|\varphi\rangle$ we

obtain the state of a qubit pair described by a vector in $\mathbb{C}^4$. To shorten the notation, very common abbreviations are used such as:

$$|\psi\rangle \otimes |\varphi\rangle \equiv |\psi\rangle|\varphi\rangle \equiv |\psi\varphi\rangle.$$

In Figure 2.2, example of a QC is depicted. It contains 3 qubits and 2 gates: $CNOT$ with $|x\rangle$ as control qubit and $|y\rangle$ as target, and $Toffoli$ with $|z\rangle$ and $|x \otimes y\rangle$ as controls and $|x\rangle$ as target.

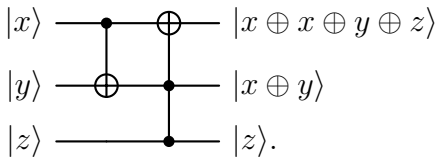


Figure 2.2: QC with 3 qubits and $CNOT$ and $Toffoli$ gates.

2.2.2 Adiabatic Quantum Computing

Adiabatic Quantum Computing (AQC) is a universal quantum computing approach [6] proposed as an application of *adiabatic theorem* to solve optimization problems [44]. The solution of a given problem corresponds to the so-called *ground state* of a quantum system with total energy described by a problem *Hamiltonian* H_P [124] on the Hilbert space where the considered quantum system is described. Within the mathematical formalism of quantum mechanics, the ground state corresponds to the eigenvector of H_P with minimum eigenvalue (if the eigenspace with lowest eigenvalue is multidimensional, the ground state is called *degenerate*). However, it is NP-hard to find minimal energy configuration of a system [124]. Thus, some initial Hamiltonian H_I with easy-to-find ground state is introduced. Note that H_I should be non-commuted with H_P.

The informal structure of an AQC is the following: the given problem is encoded into the problem Hamiltonian H_P and the quantum system is prepared in the known ground state of an initial Hamiltonian H_I. Then a time variation of the Hamiltonian from H_I to H_P is implemented according to

$$H(t) = (1 - s(t))H_I + s(t)H_P, \quad t \in [0, \tau], \tag{2.2}$$

where $s : [0, \tau] \to [0, 1]$ is a smooth monotone function such that $s(0) = 0$ and $s(\tau) = 1$. According to the adiabatic theorem, if the change is sufficiently slow then the system

remains in the instantaneous ground state with high probability. The time evolution must be slow enough to satisfy the adiabatic condition without destroying the efficiency of the QC [62]. The solution is then obtained as the ground state of the system at time τ.

2.2.3 Quantum Annealing

QA is a model of quantum computing that utilizes quantum mechanical effects to solve optimization problems [109]. QA is capable of identifying the global minimum of an objective function within a given set of potential solutions [35]. In order to be processed by QA, the problem should be transformed into a problem Hamiltonian, denoted as H_p, which mathematically describes the energy-based state of the system.

$$H_p = \sum_{i \in V} h_i {\sigma_z}^i + \sum_{(i,j) \in E} J_{ij} {\sigma_z}^i {\sigma_z}^j, \tag{2.3}$$

where ${\sigma_z}^i$ is a Pauli-Z matrix acting on qubit i, $J_{ij} = J_{ji}$ is interaction strength of the qubits i and j connected by an edge and h_i is the on-site energy of qubit i.

Like AQC, QA is driven by the motivation that a quantum system, when subjected to slow changes in its governing Hamiltonian, will stay in its ground state. The ground state of the Hamiltonian H_P corresponds to its eigenvector with the lowest eigenvalue and represents the solution to the problem. QA begins by initializing the system with the ground state of an initial Hamiltonian H_I that is relatively easy to solve. The ground state of H_I is a case when all qubits are in a superposition state of $|0\rangle$ and $|1\rangle$ [37]. QA switches from H_I to H_P sufficiently slow using the time variation from H_I to H_P (2.2).

The QA system minimizes the cost function of the problem that can be expressed in an Ising model [109]:

$$\mathcal{E}(\mathbf{z}) = \sum_{i \in V} h_i z_i + \sum_{(i,j) \in E} J_{ij} z_i z_j, \tag{2.4}$$

where $\mathbf{z} = \{-1, 1\}^{|V|}$, $G = \langle V, E \rangle$ is an undirected graph of the allowed interactions of qubits. In D-Wave systems, $h_i \in [-2, 2]$ and $J_{ij} \in [-1, 1]$, see e.g. [109] (these boundaries, however, can be extended by autoscaling).

However, there is another way to formulate a problem, and it is more natural in a sense of understanding than an Ising model. Unlike Ising model with $-1/1-$valued variables, Quadratic Unconstrained Binary Optimization (QUBO) representation acts

on 0/1−valued variables and is defined by the upper-triangular matrix Q, where Q_{ii} relates to h_i and $Q_{ij}, i \neq j$ relates to interaction strength J_{ij} in Ising formulation (2.4). QUBO objective function is as follows:

$$\mathcal{E}(\mathbf{x}) = \sum_{i \in V} Q_{ii} x_i + \sum_{(i,j) \in E} Q_{ij} x_i x_j, \tag{2.5}$$

where $\mathbf{x} = \{0,1\}^{|V|}$, $G = \langle V, E \rangle$ is an undirected graph of allowed interactions of variables.

Thus, QUBO problem is an optimization problem expressed as follows:

$$\arg\min_{\mathbf{x}} \mathbf{x}^T Q \mathbf{x}, \tag{2.6}$$

where $\mathbf{x}$ is a binary vector and Q is an upper-triangular matrix of real values.

Note that QUBO representation is equivalent to Ising one and can be converted into it using the following rule for each $i = 1, \ldots, |V|$

$$x_i = \frac{1 - z_i}{2}.$$

However, one may utilize `dimod` Python package which provides functions $ising\_to\_qubo(h, J, \ldots)$ and $qubo\_to\_ising(Q, \ldots)$ to convert Ising to QUBO and vice versa.

Note that implementation of QA implies inherent randomness - arising from the quantum superposition property — leading to the fact that the results can vary between runs. Thus, QA is considered a heuristic technique, providing an ensemble of solutions that are "good enough" rather than guaranteeing optimality. At the same time, ensemble methods may improve the performance robustness of the solutions, see e.g. [112].

2.3 State-of-the-art Quantum Circuit Optimization

In this section, we present overview of the state-of-the-art QC optimization techniques. As it was mentioned in Chapter 1, QC serves as a universal representation of quantum computation and must be synthesized for implementation on quantum hardware. However, due to the aforementioned limitations of the NISQ era, the resources required to implement the circuit must be utilized optimally. Therefore, the circuit should be optimized either during the synthesis process or independently afterwards.

By synthesis of QCs we mean the process of obtaining a (more precisely at least one) QC from a different (native to quantum hardware) representation of a quantum algorithm. The foundation of this topic was laid in [14] generalizing approach for constructing Toffoli gate from 5 two-bit gates proposed in [107] and thus building up n-bit operations with quantum single- and two-qubit gates. Further, the QC synthesis problem is described briefly in [34], where the term synthesis is defined as a compilation of the desired operator into a QC understandable for a quantum computer. In general, the synthesis process depends on the kind of the initial representation and on possible constraints and additional requirements on the target QCs. Among the synthesis processes of QCs we point out the following relevant procedures:

- **Optimization**: a synthesis process in which the target QC is required to minimize one or more cost criteria.

- **Composition**: a synthesis process from given elementary gates and/or circuits until the corresponding unitary matrix becomes equal (or approximately equal) to the specified matrix [76].

- **Decomposition**: a synthesis process whose initial representation is a given unitary matrix [36] or already a QC and the target circuit is constrained to be a composition of elementary gates from a given set.

Note that the three processes described above do not exhaust all possible synthesis processes, and a synthesis process can include more than one. Some authors adopt slightly different versions of this definition, for example reducing synthesis to decomposition.

The approximation of any unitary operator acting on n qubits with arbitrary precision ε by the so-called universal set of gates is guaranteed by the Solovay–Kitaev theorem [87] using $O(n^2 4^n \log^c(n^2 4^n/\varepsilon))$ gates. However, this is quite a large number of gates, and thus optimization is required. The general idea of QC optimization is to obtain a QC that satisfies some requirements, e.g. hardware requirements of a real machine, the set of quantum gates used for circuit synthesis and topology [31], specific connection architecture used [16, 7], depth [10] etc. As such, various optimization schemes are used, either based on the so-called search algorithms (including Monte Carlo), or using some information about the specific system, such as Bayesian optimization as in [123]. In this section, a few papers in this direction are summarized based on the criteria, models, and methods used.

2.3.1 Quantum Circuit Optimization Criteria

There are various criteria used for QC synthesis and optimization. These usually are to be minimized (smaller, better) and most of them are motivated by some specific hardware requirements (e.g. the number of gates the machine can handle) or physical requirements (e.g. decoherence time of a qubit). Hereafter we mention the most common criteria and discuss in brief their meaning accordingly.

Before enumerating the optimization criteria, recall that a QC is commonly considered as a sequence of quantum gates. For illustration, we count each of the criteria mentioned in the following for the Toffoli gate which is a universal gate from `Revlib` [117]. In QC hardware, it cannot be implemented in its initial way and that's why it is decomposed (into a sequence of elementary gates) as follows from Figure 2.3.

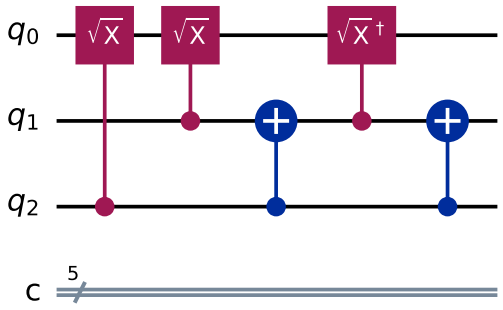


Figure 2.3: The Toffoli gate.

First, each QC has some *depth* [126, 125, 8] that indicates the longest path from the input to the output of the circuit, which is inversely proportional to the coherence time (larger, better). For the Toffoli gate depicted in Figure 2.3, *depth* equals 5.

Also, QCs can be optimized by *gate count* i.e. the number of gates in the circuit [29, 30, 54, 82, 23], including *CNOT count* i.e. the number of $CNOT$ gates in the circuit [122, 60, 31, 86, 33, 58] motivated by the fact that such gates are error-prone; *SWAP count* i.e. the number of $SWAP$ gates [16, 17, 114, 70, 129, 7, 119, 24, 23] widely used in architecture-specific (e.g. NN) circuit optimization; and *T-count* i.e. the total number of T gates or Hermitian transposes of the T gate in a quantum circuit [116, 83, 53] since such gates have significant resource cost. For the circuit demonstrated in Figure 2.3, *CNOT count* is 2, and *SWAP count* is 2, and there are no T gates.

The next optimization criteria is a *cost*, including *two-qubit cost* i.e. the number

of two-qubit gates [59, 27] and named as *the number of entanglements* in a pattern required to be applied at the beginning to produce the cluster (highly entangled) state in [56]; and *NN cost* [17, 97, 120], which is the sum of distances between gate qubits of any two-qubit gates. For the circuit demonstrated in Figure 2.3, *two-qubit cost* is 5 and *NN cost* is 2.

Note that there are QC optimization techniques based on minimization of the *number of lines* in the circuit (where ancilla lines can be used) [82]. For Toffoli gate demonstrated in Figure 2.3, there are 3 qubit lines.

For various technologies, there are several lower bounds known on the number of gates needed. In particular, the $CNOT$ count of a universal n-qubit circuit (containing only $CNOT$ and single-qubit gates) is lower bounded by [104]

$$\frac{4^n - 3n - 1}{4}.$$

The $CNOT$ count of the n-qubit Toffoli gate is at least $2n$, even if the ancillae are permitted [103].

2.3.2 Quantum Circuit Optimization Models

Most commonly, a QC is considered as a *sequence of quantum* gates [122, 123, 126, 60, 31, 125, 33, 58, 83, 8, 82] in the works reviewed. In particular, it can also be represented as a *NN circuit* [16, 17, 114, 97, 24, 23] being compliant with NN condition.

Along with circuit representation, we consider mathematical models used for optimization or synthesis. *Tree-based models* are one of the widely used ones. As an example, in [31] a circuit is studied as a tree with child nodes describing $CNOT$ positions in a circuit described by parent node. In [129], the authors consider a tree with nodes as partial permutations and edges as qubits added to partial permutation. Furthermore, a QC can be represented as a *gate dependency graph* [122, 59, 114] being directed acyclic graph with nodes being elementary gates and edges corresponding to qubits connecting these gates. Moreover, *matrix* representation [86, 116] is also utilized as well as *isometry* based representation [60, 58]. Besides that, a *SAT problem* on a Boolean function is used to encode the QC of a given depth (satisfiable if such a circuit exists) [97, 54].

2.3.3 Quantum Circuit Optimization Methods

In this section, we summarize the optimization methods used in the reviewed papers. In general, there are two basic options to synthesize optimized circuits: either by using exact method (that guarantees to find a global minimum), or by using some (meta-) heuristic approach to obtain nearly optimal solution. However, even the exact optimization method, while delivering the global minimum for the given optimization criterion, may synthesize the circuit that differs from the original (unitary operator) with a given tolerance. As such, both synthesis options are equally valuable from the point of view of solution quality.

There is a variety of optimization methods used to optimize QCs. One option is rewrite rules/templates based simplification (usually performed in a greedy way) [123, 60, 27, 97, 82, 23], in particular, enhanced with cost metric evaluation [17], or being randomized [123], or including the lines reordering [59, 70, 97, 120]. Heuristics are also commonly used in the context of QC optimization, in particular A^* search [29, 122, 31, 33, 114, 129, 30]; GP [24]; breadth-first search [86]. Also, metaheuristics are widely used to optimize QCs, such as genetic algorithms for circuit synthesis [70]; Harmony-Search with local optimization of $SWAP$ gate insertion [7]; graph traversal by ant colony [16].

Additionally, there are QC optimization approaches based on decomposition schemes [60, 58]; numerical optimization [126, 125]; Meet-in-the-Middle [83, 8]; Boolean satisfiability [114, 53, 54]. Moreover, there are large-scale approaches, including circuit partitioning [122, 126].

It is important to mention that in some papers the optimality of some circuit may be given *by design* of the corresponding synthesis method, e.g. in [54] the corresponding SAT problem gives a solution (i.e. the function is SAT), if there exists a circuit of a given (fixed) depth d, and thus optimality is obtained by solving a number of problems with increasing d.

Particular attention should be given to the open-source framework `Quantify` [89] that provides one with four-step quantitative resource analysis of QCs. More precisely, it consider a circuit via `Cirq` Google's platform and perform synthesis, optimization, transpilation, and verification of a QC. In the context of optimization, several optimization criteria are proposed (depth, $T-$depth, $CNOT$ count, T count, and number of qubit lines). As for optimization methods, they are based on template and rewrite rules (to highlight the QC's region being optimized). Moreover, this software enables verification of the circuit optimized or synthesized in order to check if the circuit is correct (metrics not selected as optimization criteria are invariant, and input-output

works as specified). The notable achievement of this software is that it allows to analyze large-scale QCs with thousands of qubits.

While conducting this overview, we present various optimization techniques for QC optimization, encompassing both exact methods and heuristics, along with different optimization models and criteria. In this thesis, we concentrate on optimizing NN circuits, highlighting that they can be optimized through multiple approaches. One method involves applying Boolean satisfiability together with heuristic A^* search to divide circuit into NN-compliant subcircuits in an optimal way (in a sense of $SWAP$ minimization) [114]. Alternatively, GP can be employed to construct optimal NN circuits by reordering the qubits [24]. Here, we base our research on these two approaches that utilize Boolean satisfiability and GP to optimize QCs. Notably, we observe that current methodologies do not utilize QA as a problem-solver in this context. Therefore, we propose to integrate QA into our ongoing research to explore its potential benefits further.

2.4 Graph Partitioning Problem

Let $G = \langle V,\ E,\ w\rangle$ be a weighted undirected graph with a set of vertices $V = \{v_1, \ldots, v_n\}$ and set $E \subseteq V \times V$ of edges $e = (v_1,\ v_2)$ with weight $w(e)$ such, that $n = |V|$ is a number of vertices and $m = |E|$ is a number of edges. The balanced k-graph partitioning ($k - GP$) problem is to find such a partition of V into k parts of equal order (i.e. with equal number of vertices) $P = \bigcup_{i=1}^{k} P_i$, minimizing the cut edge. *Cut edge* is a summary weight of the edges with end points from different partitions, i.e. for the case of 2-partition into subgraphs $G_1 = \langle V_1, E_1\rangle$ and $G_2 = \langle V_2, E_2\rangle$ cut edge is as follows and should be minimized.

$$C(V_1, V_2) = \sum_{e=(v_i,v_j)\in E,\ v_i\in V_1,\ v_j\in V_2} w(e) \to \min .$$

In Figure 2.4, graph can be bipartitioned into subgraphs $G_1 = \langle V_1,\ E_1\rangle$ with $V_1 = \{1,\ 4\}$ and $G_2 = \langle V_2,\ E_2\rangle$ with $V_2 = \{2,\ 3\}$. Cut edge is weight of edge $(1,\ 2)$ and equals 1.

GP is known to be an NP-hard problem [39]. Various approaches have been proposed to solve it. The Kernighan-Lin algorithm [66] is a well-known method that relies on the iterative swapping of pairs of vertices in order to reduce cut edge. In [65], the authors propose a technique based on multilevel recursive bisection, multilevel k-way, and multi-

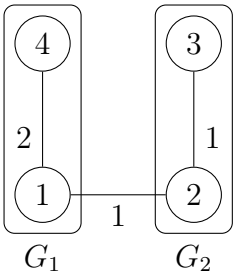


Figure 2.4: Graph to be bipartitioned.

constraint partitioning schemes, and they have developed software `metis` to perform $k-GP$. Another heuristic [38] defines the subgraph for each vertex to transfer based on the number of its neighbors in each of the subgraphs. Additionally, heterogeneous GP is proposed in [72] which involves initial balanced partitioning of computing nodes with varying workloads utilizing genetic algorithm to further reduce cut edge. Furthermore, QA has already been used to solve GP by converting it to a QUBO [113] which will be described in details further (in Section 3.2 of Chapter 3).

Chapter 3

Quantum Circuit Optimization for Linear NN Architecture by Graph Partitioning using Quantum Annealing

This chapter is a reworked version of the article [78] stretched with extra numerical experiments. Here, we investigate the problem of minimizing auxiliary $SWAP$ gates in QCs to ensure compliance with the LNN architecture. One potential solution involves the appropriate reordering of the qubit lines, specifically by finding an optimal linear arrangement on the qubit line adjacency graph [24]. Based on this, we introduce hybrid approach of optimizing QCs by its qubit line global reordering utilizing QA that provides an ensemble of suboptimal solutions. To the best of our knowledge, it is used in the context of optimizing QCs for the first time. Scalability of the proposed approach will be considered in Section 3.4.5.

3.1 Optimal Linear Arrangement by Graph Partitioning

Given an undirected weighted graph $G = \langle V, E, w \rangle$, where $V = \{v_1, \ldots, v_n\}$ is the set of vertices, $E \subseteq V \times V$ is the set of edges, and the weights of the edges are defined by the function $w : E \to \mathcal{N}$, linear arrangement problem is to find such a numbering of the vertices, $f : V \to \{1, \ldots, |V|\}$, that the total edge length given by

$$\mathcal{L}(G, f) = \sum_{\{u,v\} \in E} w(\{u, v\}) |f(u) - f(v)|, \tag{3.1}$$

is minimized [48, 41].

Linear arrangement on a weighted graph is known to be an NP-complete problem [48]. However, there are many heuristic approaches to solve it, in particular, based on the recursive balanced k-partitioning ($k-GP$) of the graph G, $k \geq 2$ [41]. In such an algorithm, the graph G is hierarchically partitioned into k subgraphs of equal order in such a way that each partition minimizes the sum of weights of the edges that are cut in order to perform the partition. Formally, let V be the set of the vertices before such a partition, and $V_1, \ldots, V_k$ be the k partitions of V, i.e. $V = V_1 \cup \cdots \cup V_k$ such that $V_i \cap V_j = \varnothing$, $i \neq j$. Then the set of edges that are cut is given as

$$E(V_1, \ldots, V_k) = \{(v_1, v_2) \in E : v_1 \in V_i, v_2 \in V_j, i \neq j\},$$

and the minimization problem is as follows:

$$\sum_{e \in E(V_1, \ldots, V_k)} w(e) \xrightarrow[V_1, \ldots, V_k]{} \min .$$

The GP is performed on the graph $\langle V, E, w \rangle$, and then recursively on each subgraph $\langle V_i, E_i, w_i \rangle$, $i = 1, \ldots, k$, where

$$E_i = \{(v_1, v_2) \in E : v_1, v_2 \in V_i\}, \quad w_i(e) = w(e), e \in E_i,$$

until $|V_i| < k$.

3.2 Quantum Circuit Optimization by Graph Partitioning

We explore the QC optimization problem for the LNN architecture. In LNN architecture, two-qubit gates only act on neighboring qubits (connected sequentially in one dimension), and $SWAP$ gates are used to relocate the non-adjacent qubits next to each other before applying a specific gate, and return them to their original positions afterwards. However, the initial arrangement (permutation) of the qubit lines within the circuit affects the required number of $SWAP$ gates (referred to as $SWAP$ count) needed to make the circuit LNN-compliant, and it is crucial to minimize their usage. Therefore, the objective is to find the (global) arrangement of the qubit lines that optimizes the circuit according to Nearest Neighbor Cost (NNC) (related to $SWAP$ count

and defined by formula (3.2)), and, as a result, minimizes the $SWAP$ count [24]. Note that such an approach optimizes only the initial placement (arrangement) of the qubit lines, i.e. the optimization is done once for a particular circuit.

First, we summarize the problem statement and the optimization approach given in [24], which is the basis of our approach. We construct the *qubit line adjacency graph* which is an undirected weighted graph $G = \langle V, E, w \rangle$, where the set V denotes the set of qubit lines in the QC and $E = \{e = \{u, v\} \subseteq V \times V\}$ denotes the set of edges between each pair of control u and target v qubits with weight $w(e) = w(\{u, v\})$ equal to the number of times this pair of qubits u and v appears in the QC. For such a graph G, we are to obtain a linear arrangement f (i.e. the order of the qubit lines in LNN topology) that minimizes the NNC of a circuit with given arrangement, which is defined using (3.1) as follows:

$$\begin{aligned} NNC(G, f) &= 2 \sum_{\{u,v\} \in E} w(\{u, v\})(|f(u) - f(v)| - 1) \\ &= 2(\mathcal{L}(G, f) - |E|). \end{aligned} \tag{3.2}$$

Thus, minimization of $NNC(G, f)$ reduces to finding an optimal linear arrangement on the qubit line adjacency graph G, e.g., by balanced GP [24] outlined in Section 2.4.

Let us intuitively explain the effect of recursive GP heuristics on the NNC of the circuit after arrangement. At each iteration of the recursion, edges (between vertices belonging to different partitions) of smaller weight are cut. Thus, edges with larger weight (corresponding to qubit lines with many gates operating on them) persist (within the same partition) until the latest iterations. As a result, the vertices corresponding to the qubit lines connected by many gates are encouraged to stay close to each other within the arrangement. That is why this heuristic approach allows one to reduce the NNC (and, as a result, $SWAP$ count) in the circuit.

The approach proposed in [24] utilizes the Divide-and-Conquer technique as suggested in [41]. The fundamental concept behind this approach involves breaking down the problem into smaller subproblems, solving them individually, and subsequently combining the solutions to obtain the solution for the original problem. However, as compared to the bipartitioning ($2 - GP$) approach suggested in [24] (i.e. at each step, each partition of the qubit line adjacency graph is to be split into two), the approach suggested in [41] is more general, and states that $k - GP$ can also be used to obtain the minimal linear arrangement of the qubit line adjacency graph. As such, we use general $k - GP$ heuristics to do the same, and it implies the novelty of current QC optimiza-

tion approach. In the next section, we use recursive decomposition of the qubit line adjacency graph into 2, 3, and 4 parts, respectively, to demonstrate feasibility of this approach.

Finally, it is worth noting that the balanced partitioning of the graph is a problem for which a QUBO formulation has already been proposed. This fact allows one to use QA-based machine, e.g. D-Wave machine, to obtain the arrangement using quantum hardware. QUBO formulation of the balanced $k-GP$ of the qubit line adjacency graph with N vertices is given in [113] as follows.

Let the binary variable $x_{ij} \in \{0, 1\}$ denote the indicator variable that the node i belongs to the part j after partition:

$$x_{ij} = \begin{cases} 1, & \text{if node } i \text{ is in part } j \\ 0, & \text{otherwise.} \end{cases}$$

Then QUBO problem is defined as follows for a QC with N qubits [113]:

$$\begin{aligned} \sum_{j=1}^{k} \mathbf{x}_j{}^T L \mathbf{x}_j + \sum_{j=1}^{k} \alpha_j \left(\sum_{i=1}^{N} x_{ij} - \frac{N}{k} \right)^2 \\ + \sum_{i=1}^{N} \gamma_i \left(\sum_{j=1}^{k} x_{ij} - 1 \right)^2 \rightarrow_{\{x_{ij}\}} \min, \end{aligned} \tag{3.3}$$

where $\mathbf{x}_j = [x_{1j}, \ldots, x_{Nj}]^T$ is a $(N \times 1)$ characteristic vector of the partition (subset) j; $\alpha_j, j = 1, \ldots, k$ and $\gamma_i, i = 1, \ldots, N$ are penalty constants [75]; and L is a Laplacian matrix with the following elements:

$$L_{ij} = \begin{cases} \sum_{k=1}^{N} w(\{i, k\}), & \text{if } i = j \\ -w(\{i, j\}), & \text{otherwise.} \end{cases}$$

The constants $\alpha_j, j = 1, \ldots, k$ penalize unequal number of nodes in the subsets after partition, and $\gamma_i, i = 1, \ldots, N$ penalize the *infeasible* solutions i.e. those containing the same node in different subsets. In general, finding the *optimal* values for these penalty constants is a separate metaoptimization problem that is not addressed here, but instead we note that the estimate of the first sum in (3.3) is a good value for α to start with [2]. We elaborate more on this subject (and find a reasonable estimate for the penalty parameters) in the next section.

After recursive performing the $k-GP$ of the qubit line adjacency graph using (3.3)

on the QA-based quantum machine (e.g. D-Wave machine), the partition tree (i.e. the tree containing the partitions as vertices in accordance with the runs of the recursion) contains the qubit line arrangement f in the order of the qubit lines contained in the leaves. By rearranging qubits within the initial circuit according to this order, a LNN-compliant circuit is obtained with an optimized NNC given by (3.2).

In summary, the suggested approach aims at optimization of the NNC (3.2) of a QC in LNN topology by global qubit arrangement which uses generalized recursive balanced partitioning heuristics of qubit line adjacency graph in QUBO formulation (3.3) on a QA machine (e.g. D-Wave machine).

Note that there is no need to construct characteristic vectors $\mathbf{x}_j$ while $2-GP$, since it is enough to define a binary vector $\mathbf{x} = \{x_i\}_{i=1}^{N}$ with $x_i = 0$ meaning that qubit i belongs to the first part and $x_i = 1$ meaning that it belongs to the second one. A simpler way of QUBO construction (alternatively to (3.3)) for the bipartitioning of an unweighted undirected graph has already been proposed [2]. Following the instructions from there, we consider several steps to build the objective function (minimizing cut edge) for bipartitioning of a weighted undirected graph. Total cut edge is defined following

$$\sum_{(u,v)\in E} w(\{u,v\})[x_u + x_v - 2x_u x_v]. \tag{3.4}$$

To perform balanced bipartitioning, the following condition should be held:

$$\sum_{u\in V} x_u = \frac{N}{2},$$

which is as follows in the form of mathematical expression:

$$\Big(\sum_{u\in V} x_u - \frac{N}{2}\Big)^2$$

simplified into

$$\sum_{u\in V} x_u(1-N) + 2\sum_{u\in V}\sum_{v>u} x_u x_v. \tag{3.5}$$

Thus, the objective function is as follows, by adding balancing constraint (3.5) multiplied by penalty constant α to (3.4):

$$\sum_{(u,v)\in E} w(\{u,v\})[x_u + x_v - 2x_u x_v] + \alpha\Big[\sum_{u\in V} x_u(1-N) + 2\sum_{u\in V}\sum_{v>u} x_u x_v\Big]. \tag{3.6}$$

By way of illustration, we present listings of qubit reordering by $2-GP$ and by $k-GP$ with $k = 3,\ 4$. Namely, for $2-GP$, we present function `solveQUBO()` that partitions the graph into 2 parts by using QA (see Algorithm 1) and function `graphPart()` that recursively bipartitions the graph until we reach leaves (see Algorithm 2). Note that we consider $\alpha_j = \gamma_i = \alpha,\ \ j = 1, \dots k,\ \ i = 1 \dots N$, as described further (in Section 3.3).

The same idea is used for $3-GP$, where the `graphPart()` function applies $k-GP$ to graph G recursively until we reach graph with vertices of count less than k. We illustrate the algorithm for $k = 3$ (see Algorithms 3 and 4).

Algorithm 1 Function `solveQUBO()` to solve $2-GP$ problem via QA

Require: Undirected weighted graph $G = \langle V, E, w\rangle$ of interactions of N qubits
Require: Penalty constant α
Ensure: $\mathbf{x} = \{x_i\}_{i=1}^{N}$ vector of binary variables indicating which partition qubit i belongs to (to the 1st partition, if x_i=0, and to the 2nd one, if x_i=1)

1: QUBO matrix $Q \leftarrow \boldsymbol{O}$
2: *Initialize Q as in* (3.6) *(lines 3-13):*
3: **for** $(u, v, w) \in E$ **do**
4: $\quad Q_{u,u} \leftarrow Q_{u,u} + w(u, v)$
5: $\quad Q_{v,v} \leftarrow Q_{v,v} + w(u, v)$
6: $\quad Q_{u,v} \leftarrow Q_{u,v} - 2w(u, v)$
7: **end for**
8: **for** $u \in V$ **do**
9: $\quad Q_{u,u} \leftarrow Q_{u,u} + \alpha(1 - N)$
10: **end for**
11: **for** $u, v \in$ pairwise combinations from V **do**
12: $\quad Q_{u,v} \leftarrow Q_{u,v} + 2\alpha$
13: **end for**
14: *Find valid solution of QUBO problem (partition should be balanced):*
15: $\mathbf{x} \leftarrow -1$ ▷ Start with invalid partition
16: **while** $\mathbf{x} = -1$ **do**
17: $\quad \mathbf{x} \leftarrow$ EmbeddingComposite(DwaveSampler()).sample_qubo(Q, num_reads = 1000) ▷ Composite to map problem to a specific topology graph
18: $\quad$ *Check solution* $\mathbf{x}$ *for validity:*
19: $\quad$ **if** $\sum_i x_i \notin [\lfloor \frac{N}{2} \rfloor, \lceil \frac{N}{2} \rceil]$ **then** ▷ Partition is unbalanced
20: $\quad\quad \mathbf{x} \leftarrow -1$ ▷ Invalid partition
21: $\quad$ **end if**
22: **end while**

Algorithm 2 Function `graphPart()` to reorder qubits by recursive $2-GP$ of the graph

Require: Undirected weighted graph $G = \langle V, E, w \rangle$ of interactions of N qubits
Require: Number of qubits to be reordered $nparts$ (order of G) ▷ Initially equal to N
Require: Permutation of qubits $total\_ordering$ ▷ Initially empty
Ensure: $\mathbf{x} = \{x_i\}_{i=1}^{N}$ vector of binary variables indicating which partition qubit i belongs to

1: **if** $nparts > 1$ **then** ▷ Partitioning should be continued
2: Initialize $\mathbf{x}$ by solving QUBO problem for G via `solveQUBO()` in Algorithm 1
3: *Initialize 2 graphs* $lG = \langle V_l, E_l, w_l \rangle$ *and* $rG = \langle V_r, E_r, w_r \rangle$ *(lines 4-18):*
4: **for** $u \in V$ **do**
5: **if** element in $\mathbf{x}$ corresponding to u is 0 **then**
6: $V_l \leftarrow V_l + u$ ▷ Add u to the first (left) graph lG
7: **else**
8: $V_r \leftarrow V_r + u$ ▷ Add u to the second (right) graph rG
9: **end if**
10: **end for**
11: **for** $(u, v, w) \in E$ **do**
12: **if** element in $\mathbf{x}$ corresponding to u is 0 and element in $\mathbf{x}$ corresponding to v is 0 **then**
13: $E_l \leftarrow E_l + (u, v, w(\{u, v\}))$ ▷ If u and v belong to lG, add edge between them to lG
14: **end if**
15: **if** element in $\mathbf{x}$ corresponding to u is 1 and element in $\mathbf{x}$ corresponding to v is 1 **then**
16: $E_r \leftarrow E_r + (u, v, w(\{u, v\}))$ ▷ If u and v belong to rG, add edge between them to rG
17: **end if**
18: **end for**
19: *Check if new graphs lG and rG need to be partitioned:* ▷ contain more than 1 vertex
20: **if** $|V_l| > 1$ **then**
21: apply $2 - GP$ to lG with $nparts \leftarrow \lfloor \frac{nparts}{2} \rfloor$ and $total\_ordering$ and obtain $left\_ordering$ ▷ Use $total\_ordering$ to store ordering obtained at previous step. We use it in the left part to consider vertices of preceding part only once.
22: $total\_ordering \leftarrow total\_ordering + left\_ordering$ ▷ Concatenate ordering obtained at previous step with ordering $left\_ordering$ obtained for lG.
23: **end if**
24: **if** $|V_r| > 1$ **then**
25: apply $2 - GP$ to rG with $nparts \leftarrow \lfloor \frac{nparts}{2} \rfloor$ and $total\_ordering \leftarrow []$ and obtain $right\_ordering$ ▷ Don't need to consider $total\_ordering$ twice (already considered for lG).
26: $total\_ordering \leftarrow total\_ordering + right\_ordering$ ▷ Concatenate ordering $right\_ordering$ obtained for rG with ordering obtained at previous step.
27: **end if**
28: **if** $|V_l| = 1$ and $|V_r| = 1$ **then** ▷ Leaves are reached, assemble ordering. **return** $V_l + V_r$
29: **end if**
30: **end if**

Algorithm 3 Function `solveQUBO()` to solve $k-GP$ via QA

Require: Undirected weighted graph $G = \langle V, E, w \rangle$ of interactions of N qubits
Require: List of degrees (summary weights of corresponding edges) *vdegree* of nodes from V
Require: Number of parts k for G to be partitioned into
Require: Penalty constant α
Ensure: $\mathbf{x} = \{x_{jN+i}\}_{i=1,\ j=0}^{i=N,\ j=k-1}$ vector of binary variables indicating which partition j qubit i belongs to

1: QUBO matrix $Q \leftarrow \boldsymbol{O}$
2: *Initialize Q as in* (3.3) *(lines 3-18):*
3: **for** $j \in [0, \ldots, k-1]$ **do**
4: **for** $u \in V$ **do**
5: $Q_{jN+u,jN+u} \leftarrow vdegree[u] + \alpha(1 - \frac{2N}{k}) - \alpha$
6: **end for**
7: **for** $(u, v, w) \in E$ **do**
8: $Q_{jN+u,jN+v} \leftarrow Q_{jN+u,jN+v} - 2w(u, v)$
9: **end for**
10: **for** $u, v \in$ pairwise combinations from V **do** ▷ Penalize unbalanced partitioning
11: $Q_{jN+u,jN+v} \leftarrow Q_{jN+u,jN+v} + 2\alpha$
12: **end for**
13: **end for**
14: **for** $i, j \in$ pairwise combinations of $[0, \ldots, k-1]$ **do** ▷ Penalize infeasible partition
15: **for** $u \in V$ **do**
16: $Q_{iN+u,jN+u} \leftarrow Q_{iN+u,jN+u} + 2\alpha$
17: **end for**
18: **end for**
19: *Find valid solution of QUBO problem (partition should be balanced and feasible):*
20: $\mathbf{x} \leftarrow -1$ ▷ Start with invalid partition
21: $onehot \leftarrow [-1] * N$ ▷ Used to check if each qubit belongs to exactly one part
22: $unbalanc \leftarrow -1$ ▷ Used to check if partition is balanced
23: **while** $\mathbf{x} = -1$ **do**
24: $\mathbf{x} \leftarrow$ EmbeddingComposite(DwaveSampler()).sample_qubo(Q, num_reads = 1000) ▷ Composite to map problem to a specific topology graph
25: *Check solution* $\mathbf{x}$ *for feasibility: (each qubit should be at only one part)*
26: **for** i in [1,. . . ,N] **do**
27: $onehot[i] \leftarrow one\_hot([x[i + jN], for\ \ j \in [0, \ldots, k-1]])$ ▷ 1, if there is exactly one 1 in the list
28: **end for**
29: **if** some element of *onehot* is not 1 **then**
30: $\mathbf{x} \leftarrow -1$ ▷ Partition is infeasible (some qubit belongs to several parts)
31: **break**
32: **end if**
33: Construct permutation *part* of qubits based on $\mathbf{x}$ as $part[i] \leftarrow j,\ \ if\ \ x[jN+i] = 1$
34: Count number of qubits $\{part\_count_j\}_{j=0}^{k-1}$ in each of k parts based on *part*
35: **if** some element of $part\_count \notin [\lfloor \frac{N}{k} \rfloor, \lceil \frac{N}{k} \rceil]$ **then** ▷ Partition is unbalanced
36: $\mathbf{x} \leftarrow -1$ ▷ Invalid partition
37: **end if**
38: **end while**

Algorithm 4 Function `graphPart()` to reorder qubits by recursive $3-GP$ of the graph

Require: Undirected weighted graph $G = \langle V, E, w \rangle$ of interactions of N qubits within the QC
Require: Number of qubits to be reordered *nparts* (order of G) ▷ Initially equal to N
Require: Permutation of qubits *total_ordering* ▷ Initially empty
Ensure: $\mathbf{x} = \{x_{jN+i}\}_{i=1,\ j=0}^{i=N,\ j=k-1}$ vector of binary variables indicating which partition j qubit i belongs to

1: **if** *nparts* > 1 **then** ▷ Partitioning should be continued
2: *vdegree* $\leftarrow$ list of summary weight of edges incident to each node in G
3: Initialize $\mathbf{x}$ by solving QUBO problem for G with $k = 3$ via `solveQUBO()` in Algorithm 3
4: *Initialize 3 graphs* $lG = \langle V_l, E_l, w_l \rangle$, $mG = \langle V_m, E_m, w_m \rangle$, *and* $rG = \langle V_r, E_r, w_r \rangle$ *(lines 5-31):*
5: **for** $j \in [0, 1, 2]$ **do**
6: **for** $u \in V$ **do**
7: **if** element in $\mathbf{x}$ corresponding to u is in j-th part **then**
8: $partition[u] \leftarrow j$ ▷ Qubit u is in j-th partition
9: **end if**
10: **end for**
11: **end for**
12: **for** $u \in V$ **do**
13: **if** $partition[u] = 0$ **then**
14: $V_l \leftarrow V_l + u$ ▷ Add u to the first (left) partition lG
15: **else if** $partition[u] = 1$ **then**
16: $V_m \leftarrow V_m + u$ ▷ Add u to the second (middle) partition mG
17: **else**
18: $V_r \leftarrow V_r + u$ ▷ Add u to the third (right) partition rG
19: **end if**
20: **end for**
21: **for** $(u, v, w) \in E$ **do**
22: **if** elements in *partition* corresponding to u and v are both 0 **then** ▷ If u and v belong to lG, add edge between them to lG
23: $E_l \leftarrow E_l + (u, v, w(\{u, v\}))$
24: **end if**
25: **if** element in *partition* corresponding to u and v are both 1 **then** ▷ If u and v belong to mG, add edge between them to mG
26: $E_m \leftarrow E_m + (u, v, w(\{u, v\}))$
27: **end if**
28: **if** element in *partition* corresponding to u and v are both 2 **then** ▷ If u and v belong to rG, add edge between them to rG
29: $E_r \leftarrow E_r + (u, v, w(\{u, v\}))$
30: **end if**
31: **end for**

Check if new graphs lG, mG, and rG need to be partitioned:
if $|V_l| > 2$ **then**
apply $3-GP$ to lG with $nparts \leftarrow \lfloor \frac{nparts}{3} \rfloor$ and $total\_ordering$ and obtain $left\_ordering$ ▷ Use $total\_ordering$ to store ordering obtained at previous step. We use it in the left part to consider vertices of preceding part only once.
$total\_ordering \leftarrow total\_ordering + left\_ordering$ ▷ Concatenate ordering obtained at previous step with ordering $left\_ordering$ obtained for lG.
end if
if $|V_m| > 2$ **then**
apply $3-GP$ to mG with $nparts \leftarrow \lfloor \frac{nparts}{3} \rfloor$ and $total\_ordering \leftarrow []$ and obtain $middle\_ordering$ ▷ Don't need to consider $total\_ordering$ twice (already considered for lG).
$total\_ordering \leftarrow total\_ordering + middle\_ordering$ ▷ Concatenate ordering $middle\_ordering$ obtained for mG with ordering obtained at previous step.
end if
if $|V_r| > 2$ **then**
apply $3-GP$ to rG with $nparts \leftarrow \lfloor \frac{nparts}{3} \rfloor$ and $total\_ordering \leftarrow []$ and obtain $right\_ordering$ ▷ Don't need to consider $total\_ordering$ twice (already considered for lG).
$total\_ordering \leftarrow total\_ordering + right\_ordering$ ▷ Concatenate ordering $right\_ordering$ obtained for rG with ordering obtained at previous step.
end if
if $|V_l| \leq 2$, and $|V_m| \leq 2$, and $|V_r| \leq 2$ **then return** $V_l + V_m + V_r$
end if
end if

For $4-GP$, the algorithm is analogous. We do not present the algorithm for the calculation of the $SWAP$ count as it was developed in [24].

3.3 Numerical Results

In this section, we set $\alpha_j = \gamma_i = \alpha,\ \ j = 1, \dots k,\ \ i = 1 \dots N$ in order to simplify the process of finding a good value for penalty constants. In such a case, the (only) penalty constant α is used to establish the balance between the objective function to minimize (the first summand) and the corresponding constraints (the second and the third summands) in the QUBO problem (3.3). Taking $\alpha > 0$ is needed to obtain solutions of the QUBO problem (3.3) (i.e. $k-GP$) that satisfy the corresponding constraints. We note that selecting optimal values of the penalty constants is a separate problem which will not be addressed here. Instead, as will be demonstrated below, the

sensitivity of the quality of the solutions obtained to the values of α is rather moderate.

To do so, we establish the following plan of numerical experiments. At first, we establish a reasonable value for penalty constant α in (3.3). We obtain it by performing series of experiments of recursive hierarchical partitioning of the qubit line adjacency graph of a (modified) circuit from [24] into $k = 2, 3, 4$ parts ($2 - GP$, $3 - GP$, and $4 - GP$, respectively) using classical counterpart of QA called Simulated Annealing (SA) implemented in `dwave-neal` package with default schedule. Replicating the runs using some sequence of values for α, we accumulate the necessary statistics and study the quality of the solutions obtained in terms of NNC. Note that SA is used due to a limited access to the QA hardware resources. Then, we run the experiments for the considered circuit using quantum D-Wave hardware (QA) and compare the results with the ones obtained by SA as well as the classical optimization software.

In this section, apart from using the NNC metric, we interchangeably use the $SWAP-$*pair count*, i.e. the number of $SWAP$ pairs, that is $NNC/2$, which is more relevant to practical implementation of the qubit routing. Indeed, each $SWAP-$pair consists of two $SWAP$ gates, one is applied before, and one after the original gate in order to make the circuit LNN-compliant.

All the experiments performed by using SA are run on local Ubuntu machine (Ubuntu 23.04 with 31 GiB RAM using 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz). To perform the experiments using QA, we use a real quantum annealer, namely, `D-Wave Advantage system 5.4` with 5614 qubits and 40050 couplers, taking the best (i.e. corresponding to minimal $SWAP$ count) out of 1000 reads. The software for providing the following experiments is available on Github via link `https://github.com/MakarMariia97/Quantum-Annealing-for-Global-QUbit-Reordering/tree/main`.

3.3.1 Sensitivity of Nearest Neighbor Cost to Penalty Constant Value

To study the sensitivity of the approach to a value of penalty constant α, we consider modification of the QC investigated in [24]. It contains 8 qubit lines. In its initial form (depicted in Figure 3.1), the circuit requires *20 $SWAP$ pairs* (i.e. NNC=40) in order to be implemented on quantum hardware with LNN architecture. The qubit line adjacency graph corresponding to the circuit is depicted in Figure 3.2.

Note that in order to perform a balanced GP, number of qubit lines N within the circuit should be equal to the power of k. However, due to the limited capabilities of

D-Wave, we decide to increase N to one of the following values (depends on which is closer to N): the smallest power of k, that is greater than N; or double of the greatest power of k, that is smaller than N. These extra qubit lines will be omitted in the final permutation. Thus, we increase $N = 8$ to 9 for the considered circuit to perform its $3 - GP$; for $2 - GP$ and $4 - GP$ there is no need to add some extra qubit lines.

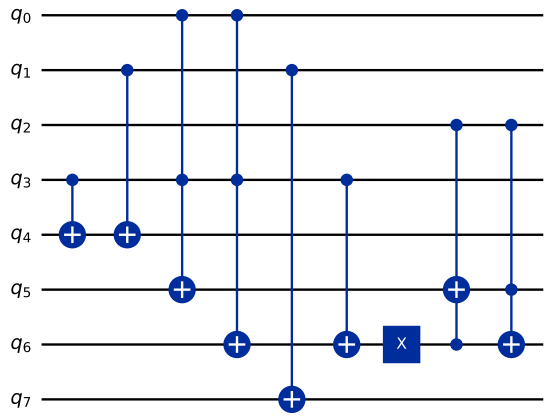


Figure 3.1: Modified circuit from [24]. $SWAP-$pair count required to make it LNN-compliant is 20. (The first $CNOT$ already works on adjacent qubits q_3 and q_4. For the second $CNOT$, 2 $SWAP-$pairs are required: 2 $SWAP$s are needed to make q_1 and q_4 adjacent before the operation and two $SWAP$s are needed to restore the qubits placement. The same logic is applied for the rest of the gates.)

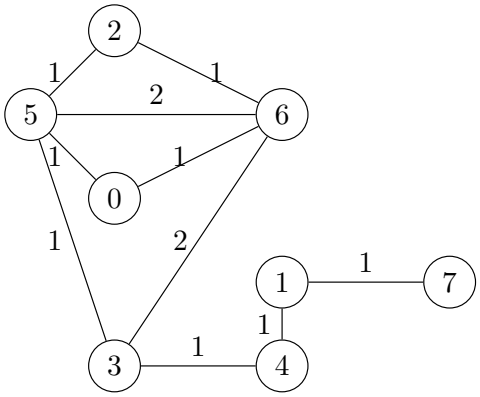


Figure 3.2: Qubit line adjacency graph of the circuit from Figure 3.1.

To establish some estimate on the number of $SWAP$ pairs in the chain after optimization, we apply the qubit line adjacency graph bipartitioning heuristics suggested in [24] to the graph given in Figure 3.2 using classical software, namely, the `metis` package and `partition()` of `dwave-networkx` Python package with a constrained quadratic

model sampler `ExactCQMSolver()`. These software packages for GP are based on a deterministic approach. The solution returned by the former package is a circuit that requires *10 $SWAP-$pairs* to be LNN-compliant, while the latter package gives a circuit that requires *12 $SWAP-$pairs*.

To establish sensitivity of the approach presented in Section 3.2 to the penalty constant α, the hierarchical $k-GP$ (using QUBO formulation) is performed for an even sequence $\alpha = 2, 4, \ldots, 20$, using SA sampler until 10^4 valid (correct) arrangements are obtained. An empirical distribution of the of the $SWAP-$pair count is built for each value of penalty constant α. The values of α were selected in a neighborhood of the number of $SWAP-$pairs obtained by the classical packages. Note that we omit $\alpha = 1$ due to empirical evidence of an insufficient penalty size. Indeed, it turned out in the experiments that for such α the algorithm solves the QUBO problem successfully rather infrequently (about 2 valid runs out of 10^4), which results in around 29 days needed to obtain the necessary sample size (since one run takes about 8.5 minutes for $4-GP$). One possible way to speed up the computations is based on using parallel computing. However, we don't address this case here.

For illustrative purposes, we demonstrate distribution of $SWAP-$pair count of the considered circuit only after applying recursive $2-GP$. From Figure 3.3, we see that median of $SWAP-$pair count increases when α increases. In this figure, there are outliers for $\alpha > 4$, and they mean solutions more extreme than expected variation (represented by 'whiskers'). E.g., for $\alpha = 6$ outliers demonstrate (rare) suboptimal solutions resulting in more than 25 $SWAP-$ pairs, while the median solution is around 12 $SWAP-$ pairs. On the contrast, for $\alpha = 16$ outliers indicate that there are solutions better than the median one. As we will see further (in Section 3.3.2), optimization algorithm terminates unsuccessfully for more than a half of trials for $\alpha = 2$. For this reason this case is omitted in Figure 3.4, where we see that $SWAP-$pair count distribution becomes right-skewed when α decreases. The same behavior is observed for recursive $3-GP$ and $4-GP$.

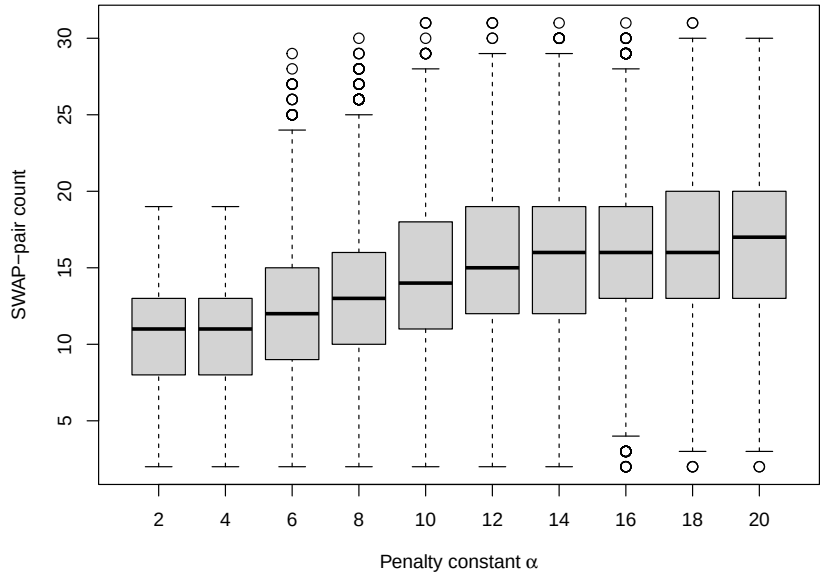


Figure 3.3: "Box-and-whisker" plot of $SWAP-$pair count obtained for hierarchical $2-GP$ for different values of α. Each sample contains 10^4 results (arrangements of qubit lines) obtained by solving the $2-GP$ problem (3.3) using SA sampler.

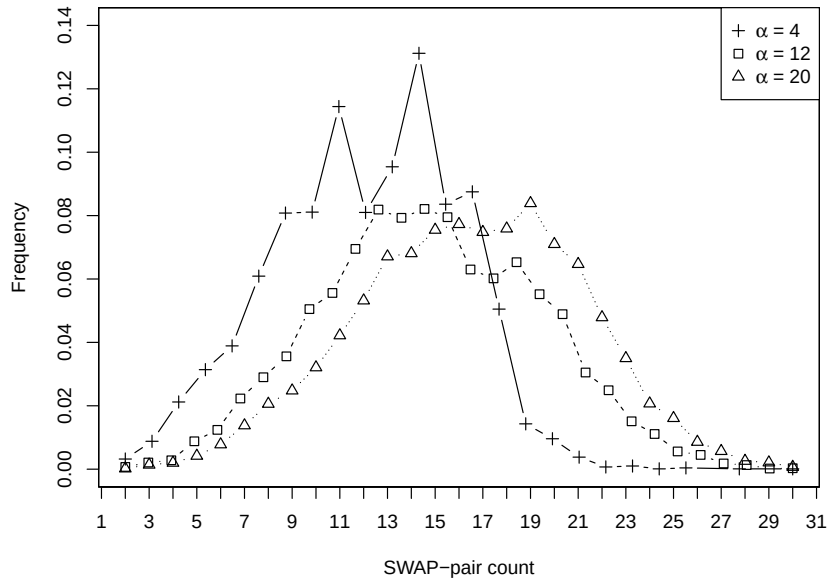


Figure 3.4: Distribution of $SWAP-$pair count in (feasible) circuits obtained by running hierarchical $2-GP$ on a SA sampler for various values of α.

3.3.2 Sensitivity of Unsuccessful Trials Count to Penalty Constant Value

In this section, we study the sensitivity of unsuccessful trials count to a value of penalty constant α. An *unsuccessful trial* is a situation when the algorithm rejects the graph partition that does not satisfy the constraints of the problem (3.3) (i.e. the partition is done in such a way that some vertex belongs to several parts or partitions are of unequal order). A large number of unsuccessful trials may indicate that α is not large enough to penalize the unacceptable solutions. The number of these unsuccessful trials are counted for different values of α for $2-GP$, $3-GP$, and $4-GP$ obtained by running the algorithm using SA.

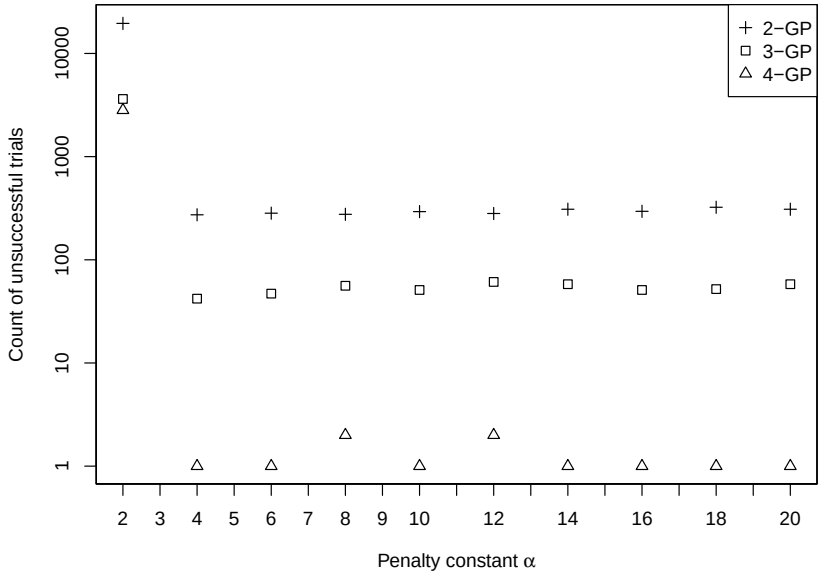


Figure 3.5: Comparison for distribution of unsuccessful trials with different α. Logarithmic scale of Y-axis.

As we see from Figure 3.5, for $2-GP$, $3-GP$, and $4-GP$ the largest count of unsuccessful trials is for $\alpha = 2$ and it is relatively small for $\alpha \geq 4$. Among the versions of GP, the smallest count of unsuccessful trials is obtained by $4-GP$.

3.3.3 Sensitivity of Metric of Success to Penalty Constant Value

In this section, we compare the $2-GP$, $3-GP$, and $4-GP$ for different values of α using the *metric of success* (MoS) defined as the probability to obtain $SWAP-$pair

count less than or equal to some threshold. Note that this metric is the cumulative distribution function of $SWAP-$pair count, i.e. $F(m) = P(SWAP-\text{pair count} \leq m)$ for some threshold m. We observe a MoS for $2-GP$, $3-GP$, and $4-GP$ by running the algorithm using SA and study if there is some dependence of this metric on α.

Recall that the classical solution obtained by `metis` software returns a circuit that requires *10 $SWAP-$pairs*. So, $m = 9$ will be the first threshold in order to estimate the probability to outperform classical solution. Also, we would like to assess the probability to obtain minimum practical $SWAP-$pair count for $2-GP$, $3-GP$, and $4-GP$. For $2-GP$ and $3-GP$ the minimal $SWAP-$pair count experimentally obtained (using SA) is 2 and for $4-GP$ it is 3 at best. The MoS is estimated for these two thresholds for an even sequence $\alpha = 2, 4, \ldots, 20$ by running the algorithm for 10^4 times.

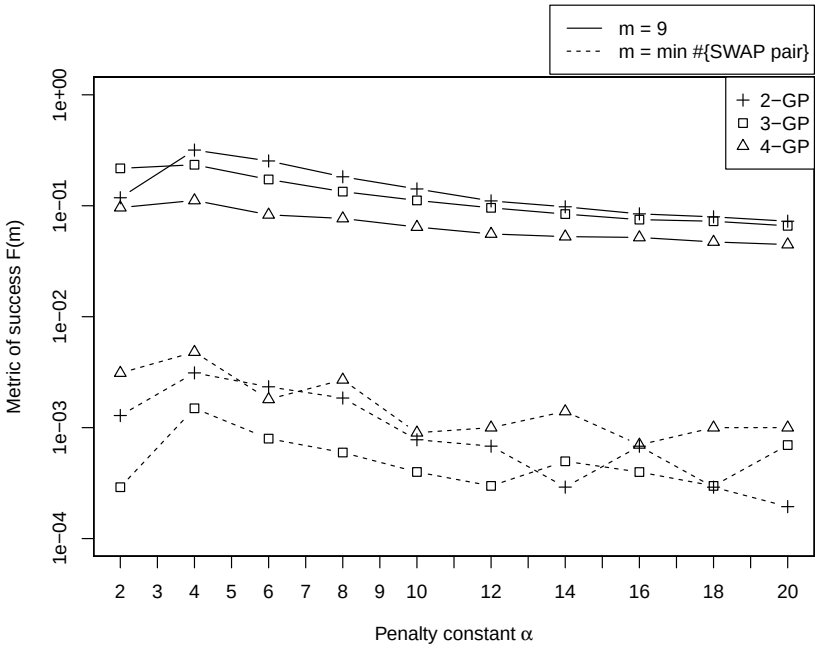


Figure 3.6: Comparison of MoS with different α. Note the logarithmic scale for the MoS.

In Figure 3.6, we couple points with lines for visual clarity. It seems that: 1) there is no dependence of the probability to obtain minimum $SWAP$ count on α, 2) the probability to obtain better solution than classical one depends on α exponentially (note the logarithmic scale on the vertical axis). The value $\alpha = 4$ provides a relatively high MoS for all the three versions of partitioning, with the largest value 0.317 attained for $2-GP$.

3.3.4 Experiments using D-Wave Machine

Based on the sensitivity studies in Sections 3.3.2 and 3.3.3, the penalty constant $\alpha = 4$ provides reasonably good results: relatively high MoS and small count of unsuccessful trials. By this reason, $\alpha = 4$ is used hereafter to optimize the QCs using D-Wave machine. We note that D-Wave machine provides us with limited access time and running of $2-GP$, $3-GP$, and $4-GP$ takes different time. To make experimental results comparable, we propose to run 3 equal (in runtime) periods of experiments: one to each of the three GP versions. Accordingly, the equal runtime was sufficient to have 84 runs of the $2-GP$, 114 runs of the $3-GP$, and 390 runs of the $4-GP$ (this difference is due to the smaller number of iterations used to obtain GP with larger number of components).

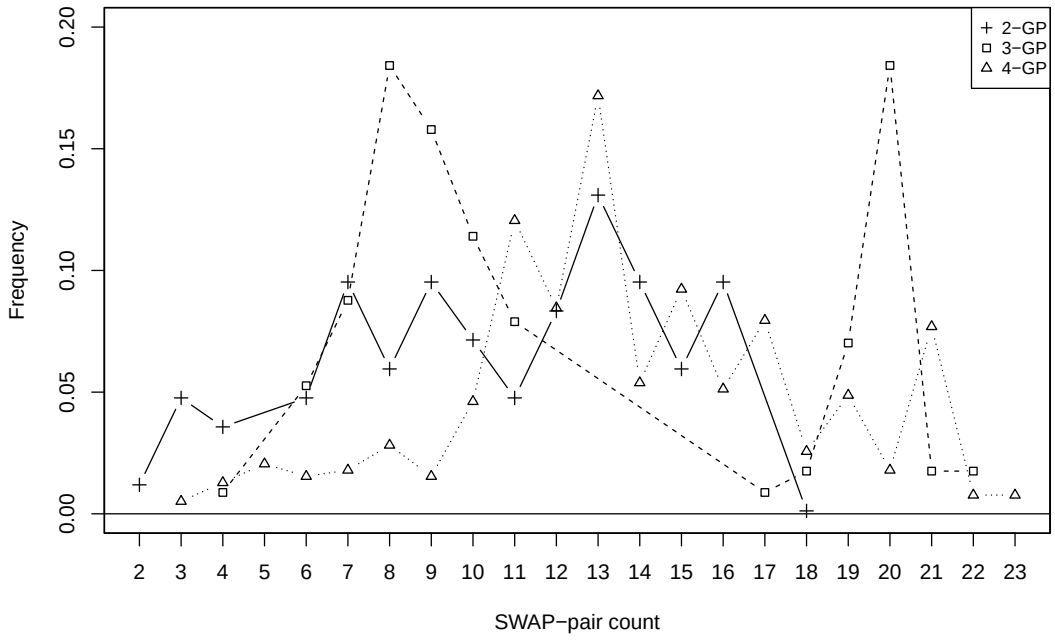


Figure 3.7: Comparison of $SWAP-$pair count PDF of $2-GP$, $3-GP$, and $4-GP$ for $\alpha = 4$ obtained on D-Wave machine. Initial $SWAP-$pair count is 20.

Figure 3.7 depicts $SWAP-$pair count obtained on D-Wave machine for all the three versions of GP. For visual clarity, we omit points with zero probability. According to Figure 3.7, $2-GP$ implemented on D-Wave machine always returns result better than the initial $SWAP-$pair count (equal to 20) and the best result is better than ones returned by $3-GP$ and $4-GP$. Also, mode of $SWAP-$pair count probability density

function (PDF) obtained by $3-GP$ is less than modes returned by $2-GP$ and $4-GP$.

Table 3.1 illustrates the comparison of the results obtained by applying the proposed approach and solving the constructed GP problem using QA with the results obtained by SA and classically (using `metis` and `partition()` function (performing $k-GP$) of `dwave-networkx` Python package separately) for $2-GP$, $3-GP$, and $4-GP$. As we see, QA returns worse solution (as opposed to SA) only for $3-GP$ and it always significantly outperforms the two other classical solvers. Supremacy of SA could be explained by the fact that both SA and QA are heuristic techniques with probability-based acceptance of suboptimal solutions. The number of runs of the algorithm on QA during the limited access time for all the three versions is also given. Moreover, we calculate the ratio between MoS for the threshold $m = 9$ obtained by QA and SA, $F_{QA}(9)/F_{SA}(9)$, to assess if QA optimizes the circuit more frequently. The best solution (obtained by QA) is illustrated in Figure 3.8.

Table 3.1: Comparison of the optimization results obtained by QA and classically (using SA, `metis`, and `partition()` function of `dwave-networkx`) for the circuit from Figure 3.1. Note that we use `metis` to perform bipartitioning only.

k	Number of runs on QA	$\frac{F_{QA}(9)}{F_{SA}(9)}$	QA ($SWAP-$ pair count)	SA ($SWAP-$ pair count)	`metis` ($SWAP-$ pair count)	`partition()` ($SWAP-$ pair count)
2	84	1.18	2	2	10	12
3	114	1.81	4	2	-	20
4	390	1	3	3	-	13

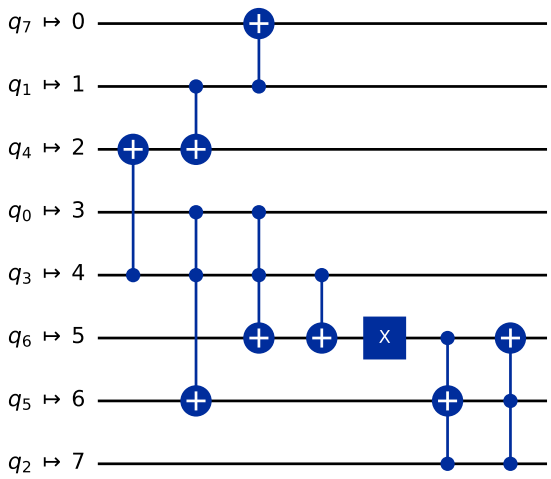


Figure 3.8: Circuit from Figure 3.1 optimized by 2−GP for $\alpha = 4$ on D-Wave machine. After optimization, qubits are placed as follows: q_7 is the first one, followed by q_1 after which q_4 is placed. Qubit q_0 follows q_4 and precedes q_3 after which q_6 is placed. Qubit q_5 follows q_6, and q_2 takes last place. In other words, qubit placement is depicted by arrows in the left of the figure and is $(q_7, q_1, q_4, q_0, q_3, q_6, q_5, q_2)$, resulting in 2 $SWAP-$pairs required to make the QC LNN-compliant.

3.4 Further Validation

In this section, we further validate our approach. We select four QCs from several benchmark libraries, each differing from the circuit considered in Section 3.3 by the number of qubits and gates. The experiments are performed by QA using the penalty constant $\alpha = 4$ selected in Section 3.3.4. The comparison of QA results is done to the classical optimization methods only. Scalability of the proposed approach in terms of the number of (physical/logical) qubits is discussed at the end of this section.

3.4.1 Double Toffoli Gate Optimization

Hereafter we perform an optimization of the double Toffoli gate (circuit composed of two Toffoli gates in a sequence) presented in RevLib [117]. Double Toffoli gate requires 5 $SWAP$ pairs initially, that is illustrated in Figure 3.9. This is because the first two gates (and, as a consequence, the fourth and the last ones) operated on adjacent qubits, whereas the third gate (and the fifth one) acts on qubits q_0 and q_3, and 2 $SWAP$s are used before this operation to make this pair of qubits adjacent, and 2 $SWAP$s are used after the operation to restore initial qubit placement. The sixth gate operates on qubits

q_1 and q_3, and 1 $SWAP$ is used before this gate and after it. The qubit line adjacency graph of the circuit is as given in Figure 3.10.

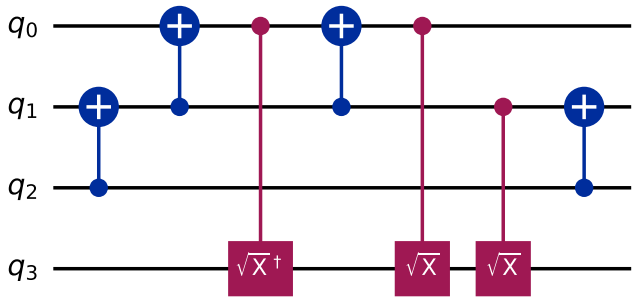


Figure 3.9: Double Toffoli gate. $SWAP-$pair count required to make it LNN-compliant is 5.

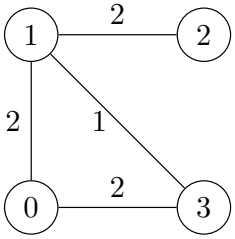


Figure 3.10: Qubit line adjacency graph of the circuit from Figure 3.9.

Note that the number of qubits $N = 4$, so we need to add 2 extra qubits to perform $3-GP$. For $2-GP$ and $4-GP$ we do not need some extra qubits.

Table 3.2 illustrates the comparison of the results obtained by applying the proposed approach and solving the GP problem using QA to the results obtained classically (using `metis` and `partition()` function of `dwave-networkx` Python package separately) for $2-GP$, $3-GP$, and $4-GP$. As we see, QA outperforms classical methods in minimization of $SWAP-$pair count for $3-$ and $4-GP$, whereas performance on $2-GP$ is comparable. In Figure 3.11, the best solution (obtained by QA) is illustrated. Circuit obtained by $3-GP$ (by QA) is depicted in Figure 3.12.

Table 3.2: Comparison of the optimization results obtained by QA and classically (using `metis`, and `partition()` function of `dwave-networkx`) for Double Toffoli in Figure 3.9. Recall that we use `metis` to perform bipartitioning only.

k	QA ($SWAP-$pair count)	`metis` ($SWAP-$pair count)	`partition()` ($SWAP-$pair count)
2	1	1	1
3	2	-	3
4	1	-	3

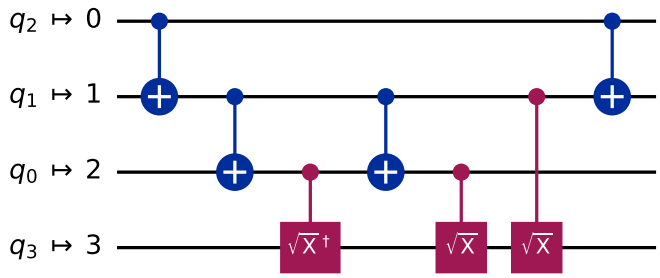


Figure 3.11: Double Toffoli gate after $2-$GP (and $4-$GP) on D-Wave machine. After optimization, qubits are placed as follows: q_2 is the first one, followed by q_1 after which q_0 is placed. Qubit q_3 takes last place. In other words, qubit placement is (q_2, q_1, q_0, q_3) (as depicted by arrows in the left), resulting in 1 $SWAP-$pair required to make the QC LNN-compliant.

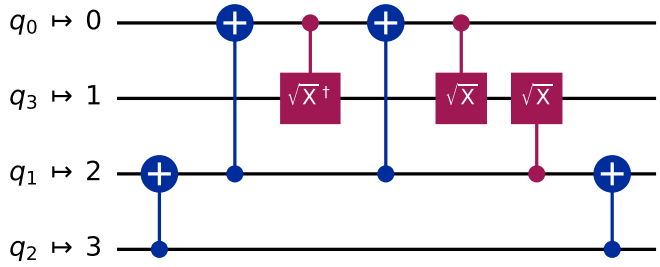


Figure 3.12: Double Toffoli gate after $3-$GP on D-Wave machine. After optimization, qubits are placed as follows: q_0 is the first one, followed by q_3 after which q_1 is placed. Qubit q_2 takes last place. In other words, qubit placement is (q_0, q_3, q_1, q_2), resulting in 2 $SWAP-$pairs required to make the QC LNN-compliant.

3.4.2 Optimizing Gate implementing Multiplication of Two Elements of Galois Field GF(2^4)

Now, we consider the circuit that performs multiplication of two elements of a Galois Field GF(2^4) (the so-called GF(2^4) multiplier). This circuit is presented in the Reversible logic synthesis benchmark page by D. Maslov [80] and is demonstrated in Figure 3.13. Initially, it requires 96 $SWAP$ pairs. The first gate acts on qubits q_3, q_5, and q_8, and 3 $SWAP$s are used before this gate to make these qubits adjacent, and 3 $SWAP$s are used to restore initial order after this gate. Analogous logic is applied to other gates included in this QC. The qubit line adjacency graph corresponding to the considered circuit is as depicted in Figure 3.14.

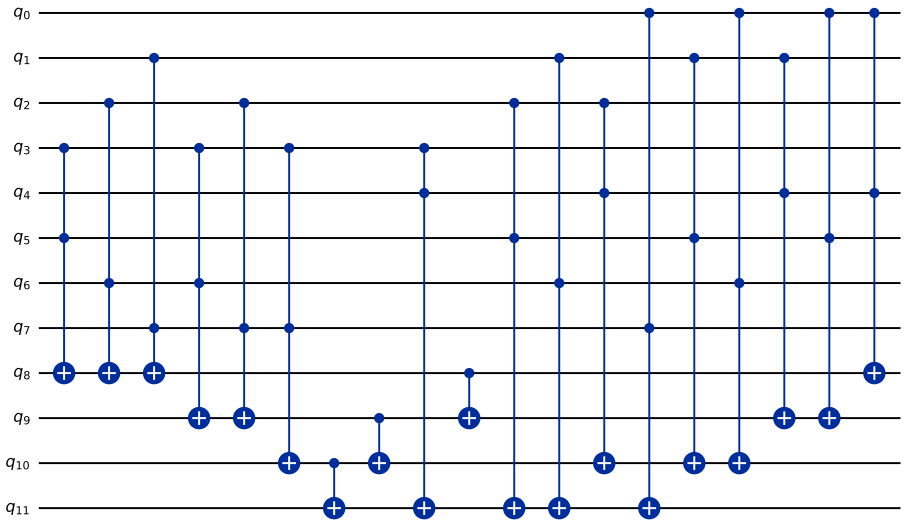


Figure 3.13: GF(2^4) multiplier. $SWAP-$pair count required to make it LNN-compliant is 96.

Note, that number of qubits $N = 12$, so we need to add 4 extra qubits to perform $2-GP$ and $4-GP$, and 6 extra qubits to perform $3-GP$.

Table 3.3 illustrates the comparison of the results obtained by applying the proposed approach and solving the constructed GP problem using QA to the results obtained classically (using `metis` and `partition()` function of `dwave-networkx` Python package separately) for $2-GP$, $3-GP$, and $4-GP$. As we see, performance of QA is better or equal to that of classical algorithm (`partition()`). In Figure 3.15, the best solution (obtained by QA) is illustrated. Circuits obtained by $2-GP$ and $3-GP$ (by QA) are depicted in Figure 3.16 and Figure 3.17, respectively.

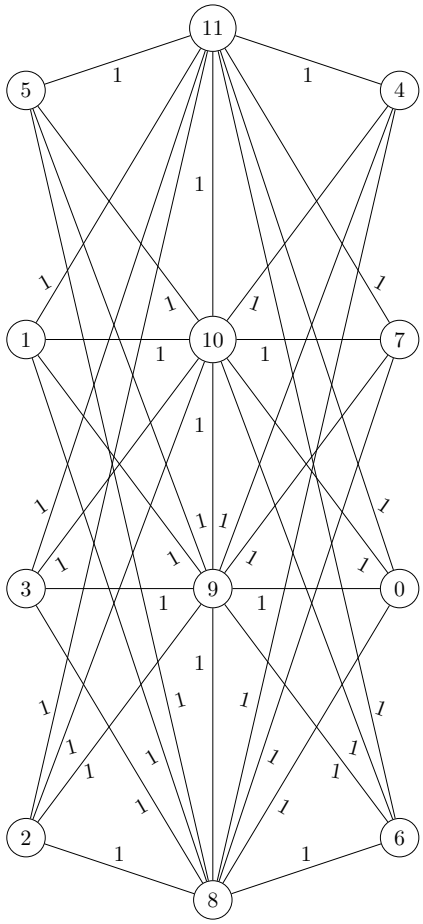


Figure 3.14: Qubit line adjacency graph of the circuit from Figure 3.13.

Table 3.3: Comparison of the optimization results obtained by QA and classically (using `metis` for bipartitioning, and `partition()` function of `dwave-networkx`) for GF(2^4) multiplier in Figure 3.13. Note that result for 4-GP by `partition()` tool is missing due to software issue.

k	QA ($SWAP$−pair count)	`metis` ($SWAP$−pair count)	`partition()` ($SWAP$−pair count)
2	75	81	80
3	78	-	78
4	71	-	-

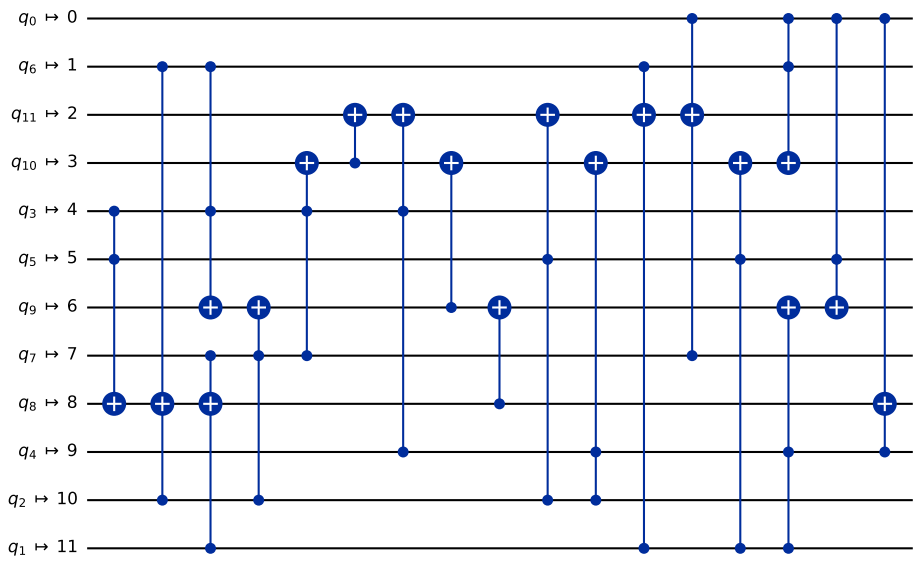


Figure 3.15: GF(2^4) multiplier optimized by 4−GP on D-Wave machine. After optimization, qubits are placed as follows: q_0 is the first one, followed by q_6 after which q_{11} is placed. Qubit q_{10} follows q_{11}. Qubit q_1 takes last place. Placement of other qubits is depicted by arrows in the left of the figure, and is $(q_0, q_6, q_{11}, q_{10}, q_3, q_5, q_9, q_7, q_8, q_4, q_2, q_1)$, resulting in 71 $SWAP$−pairs required to make the QC LNN-compliant.

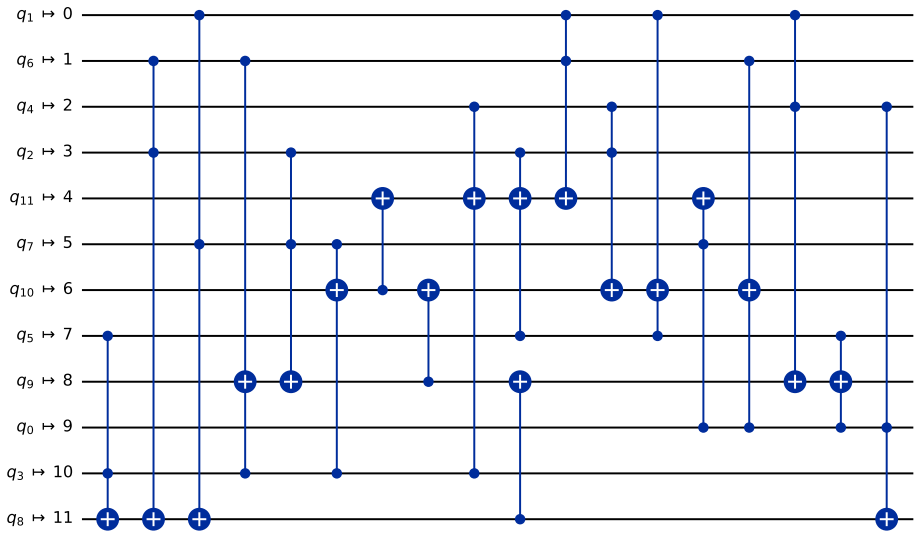


Figure 3.16: GF(2^4) multiplier optimized by 2−GP on D-Wave machine. After optimization, qubits are placed as follows: q_1 is the first one, followed by q_6 after which q_4 is placed. Qubit q_2 follows q_4. Qubit q_8 takes last place. Placement of other qubits is depicted by arrows in the left of the figure, and is $(q_1, q_6, q_4, q_2, q_{11}, q_7, q_{10}, q_5, q_9, q_0, q_3, q_8)$, resulting in 75 $SWAP$−pairs required to make the QC LNN-compliant.

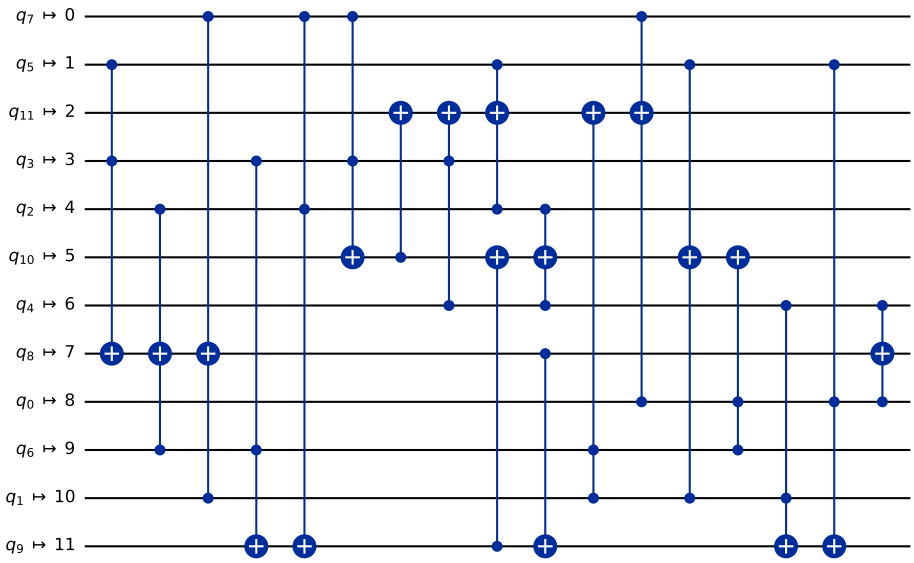


Figure 3.17: GF(2^4) multiplier optimized by 3−GP on D-Wave machine. After optimization, qubits are placed as follows: q_7 is the first one, followed by q_5 after which q_{11} is placed. Qubit q_3 follows q_{11}. Qubit q_9 takes last place. Placement of other qubits is depicted by arrows in the left of the figure, and is $(q_7, q_5, q_{11}, q_3, q_2, q_{10}, q_4, q_8, q_0, q_6, q_1, q_9)$, resulting in 78 $SWAP$−pairs required to make the QC LNN-compliant.

3.4.3 Hamming Gate Optimization

Here, we perform an optimization of the Hamming gate constructed in NCT-gate library from RevLib [117]. Hamming gate requires 32 $SWAP$ pairs initially, that is illustrated in Figure 3.18. As seen from this figure, the first gate works on adjacent qubits q_4 and q_5, whereas the second gate acts on qubits q_1 and q_6, and 4 $SWAP$−pairs are used to make these qubits adjacent before and after the gate (totally). Analogous logic is applied to other gates included in the QC. The qubit line adjacency graph of the circuit is as given in Figure 3.19.

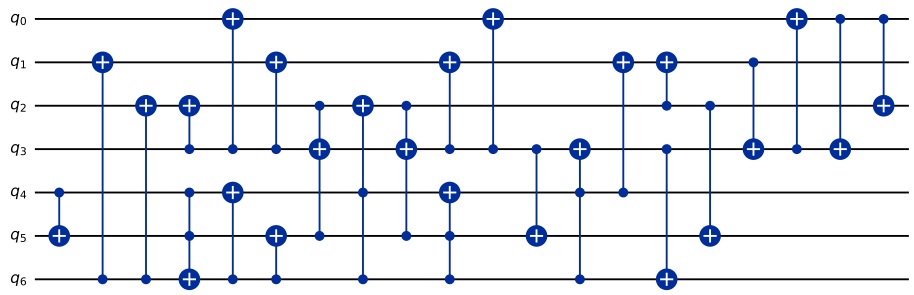


Figure 3.18: Hamming gate. $SWAP-$pair count required to make it LNN-compliant is 32.

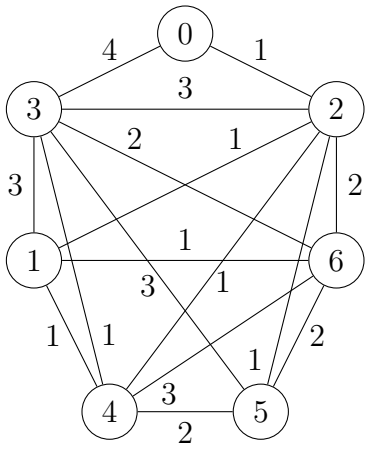


Figure 3.19: Qubit line adjacency graph of the circuit from Figure 3.18.

Note, that the number of qubits $N = 7$, so we need to add an extra qubit to perform $2 - GP$ and $4 - GP$, and 2 extra qubits to perform $3 - GP$.

Table 3.4 illustrates the comparison of the results obtained by applying the proposed approach and solving the GP problem using QA to the results obtained classically (using `metis` and `partition()` function of `dwave-networkx` Python package separately) for $2 - GP$, $3 - GP$, and $4 - GP$. As we see, QA outperforms classical methods in minimization of $SWAP-$pair count for all the three versions of GP. In Figure 3.20 and Figure 3.21, the best solutions (obtained by QA) are illustrated. Circuit obtained by $4 - GP$ (QA) is depicted in Figure 3.22.

Table 3.4: Comparison of the optimization results obtained by QA and classically (using `metis`, and `partition()` function of `dwave-networkx`) for the circuit from Figure 3.18. Note that we use `metis` to perform bipartitioning only.

k	QA ($SWAP-$pair count)	`metis` ($SWAP-$pair count)	`partition()` ($SWAP-$pair count)
2	29	31	37
3	29	-	32
4	30	-	37

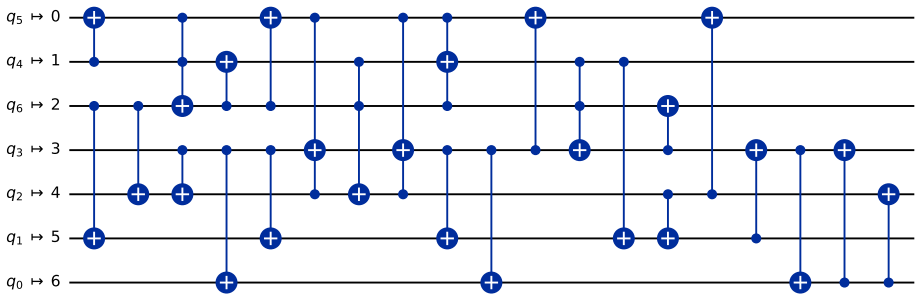


Figure 3.20: Hamming gate after $2-$GP on D-Wave machine. After optimization, qubits are placed as follows: q_5 is the first one, followed by q_4 after which q_6 is placed. Qubit q_0 takes last place. Placement of other qubits is depicted by arrows in the left of the figure, and is $(q_5, q_4, q_6, q_3, q_2, q_1, q_0)$, resulting in 29 $SWAP-$pairs required to make the QC LNN-compliant.

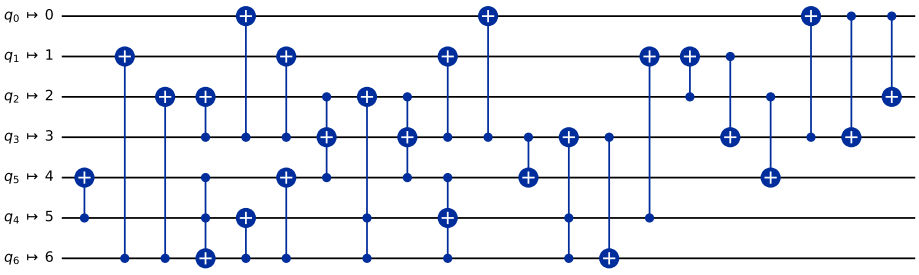


Figure 3.21: Hamming gate after $3-$GP on D-Wave machine. After optimization, qubits are placed as follows: q_0 is the first one, followed by q_1 after which q_2 is placed. Qubit q_6 takes last place. Placement of other qubits is depicted by arrows in the left of the figure, and is $(q_0, q_1, q_2, q_3, q_5, q_4, q_6)$, resulting in 29 $SWAP-$pairs required to make the QC LNN-compliant.

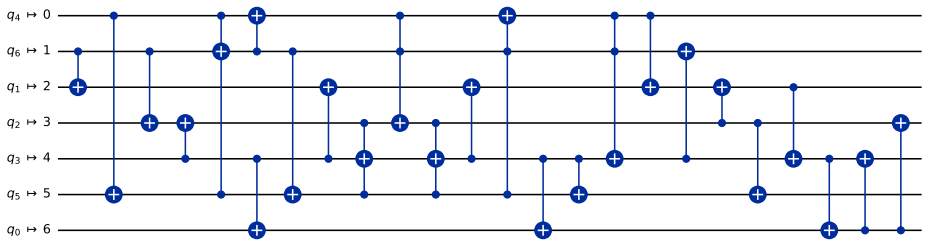


Figure 3.22: Hamming gate after 4−GP on D-Wave machine. After optimization, qubits are placed as follows: q_4 is the first one, followed by q_6 after which q_1 is placed. Qubit q_0 takes last place. Placement of other qubits is depicted by arrows in the left of the figure, and is $(q_4, q_6, q_1, q_2, q_3, q_5, q_0)$, resulting in 30 $SWAP-$pairs required to make the QC LNN-compliant.

3.4.4 2-to-4 Decoder Gate Optimization

Here, we perform an optimization of the 2-to-4 decoder constructed in the library of elementary quantum gates and presented in RevLib [117]. The gate requires 6 $SWAP$ pairs initially, that is illustrated in Figure 3.23. The first gate acts on adjacent qubits q_1 and q_2, whereas the second gate operated on qubits q_1 and q_3 that become adjacent after swapping, e.g. q_1 and q_2. The third gate also works on non-adjacent qubits, and we use 2 $SWAP-$pairs before and after this gate (totally). Analogous logic is applied to other gates included in the QC. The qubit line adjacency graph of the circuit is as given in Figure 3.24.

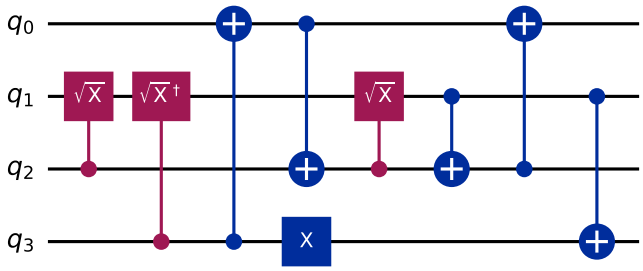


Figure 3.23: 2-to-4 decoder gate. $SWAP-$pair count required to make it LNN-compliant is 6.

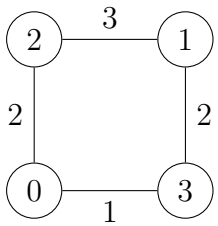


Figure 3.24: Qubit line adjacency graph of the circuit from Figure 3.23.

Note that the number of qubits $N = 4$, so we need to add 2 extra qubits to perform $3 - GP$. For $2 - GP$ and $4 - GP$ we do not need some extra qubits.

Table 3.5 illustrates the comparison of the results obtained by applying the proposed approach and solving the GP problem using QA to the results obtained classically (using `metis` and `partition()` function of `dwave-networkx` Python package separately) for $2 - GP$, $3 - GP$, and $4 - GP$. As we see, QA outperforms classical methods in minimization of $SWAP-$pair count for $2 - GP$ and $4 - GP$. In Figure 3.25, the best solution (obtained by QA) is illustrated. Circuits obtained by $3 - GP$ and 4−GP (by QA) are depicted in Figure 3.26 and Figure 3.27, respectively.

Table 3.5: Comparison of the optimization results obtained by QA and classically (using `metis`, and `partition()` function of `dwave-networkx`) for the circuit from Figure 3.23. Note that we use `metis` to perform bipartitioning only.

k	QA ($SWAP-$pair count)	`metis` ($SWAP-$pair count)	`partition()` ($SWAP-$pair count)
2	2	4	6
3	4	-	2
4	4	-	6

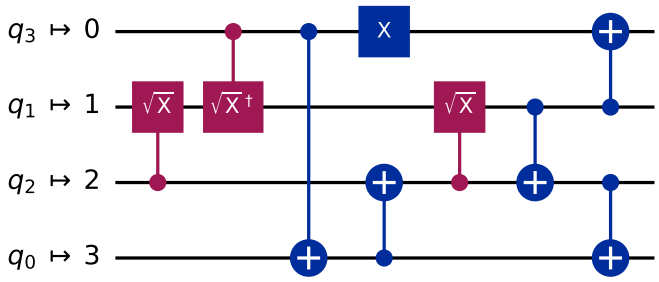


Figure 3.25: 2-to-4 decoder optimized via 2−GP by D-Wave. After optimization, qubits are placed as follows: q_3 is the first one, followed by q_1 after which q_2 is placed. Qubit q_0 takes last place. Qubit placement (depicted in the left of the figure) is (q_3, q_1, q_2, q_0), resulting in 2 $SWAP-$pairs required to make the QC LNN-compliant.

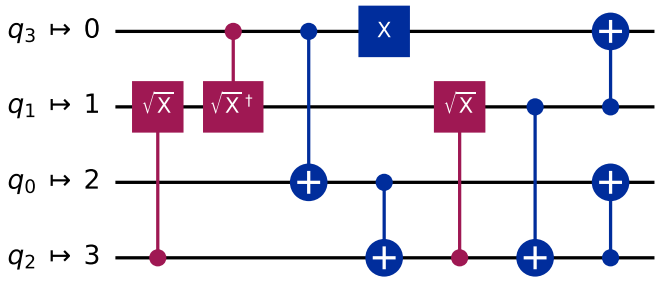


Figure 3.26: 2-to-4 decoder optimized via 3−GP by D-Wave. After optimization, qubits are placed as follows: q_3 is the first one, followed by q_1 after which q_0 is placed. Qubit q_2 takes last place. Qubit placement (depicted in the left of the figure) is (q_3, q_1, q_0, q_2), resulting in 4 $SWAP-$pairs required to make the QC LNN-compliant.

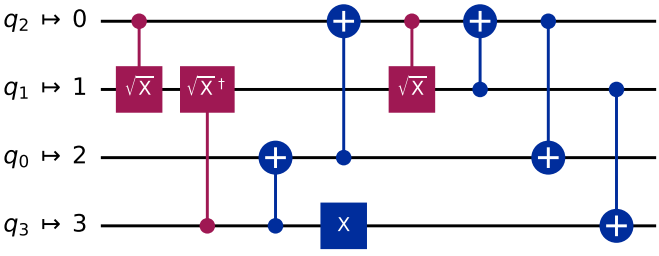


Figure 3.27: 2-to-4 decoder optimized via 4−GP by D-Wave. After optimization, qubits are placed as follows: q_2 is the first one, followed by q_1 after which q_0 is placed. Qubit q_3 takes last place. Qubit placement (depicted in the left of the figure) is (q_2, q_1, q_0, q_3), resulting in 4 $SWAP-$pairs required to make the QC LNN-compliant.

3.4.5 Evaluating Overhead of the Proposed Approach

As was mentioned earlier, an optimization problem is usually stated as a QUBO problem to be solved by QA on D-Wave. The QUBO format represents the problem as a set of logical qubits and couplers. When QUBO is solved on quantum hardware, one needs to embed it into the qubit connectivity topology [121, 3] which is rather sparse. This procedure is called minor embedding. It converts logical problems with structure that does not correspond to the hardware graph to be programmed onto the hardware [93]. In QA, the minor embedding represents each variable in the logical problem by a combination of interconnected physical qubits. The dimension of the optimization problem that can be solved on quantum annealers is restricted. In particular, for `Advantage` quantum annealer with more than 5000 physical qubits and 15-way qubit connectivity, the upper bound on the number of logical qubits was established as 180 [1].

Table 3.6: Numerical results of the maximum memory requirements presented as a pair of number of logical and physical (in parentheses) qubits required to optimize QCs from Section 3.3.4, and Sections 3.4.1–3.4.4 by QA.

k	Modified circuit from [24] ($N = 8$)	double Toffoli ($N = 4$)	GF(2^4) multiplier ($N = 12$)	Hamming gate ($N = 7$)	2-to-4 decoder ($N = 4$)
2	8 (12)	4 (4)	16 (39)	8 (12)	4 (4)
3	27 (64)	18 (32)	54 (233)	27 (66)	18 (30)
4	32 (90)	16 (29)	64 (356)	32 (95)	16 (28)

In Table 3.6, we present the dimensions of the QUBO problem associated with the quantum circuit to be optimized, represented as a pair indicating the number of logical qubits and physical qubits (shown in parentheses). This pair reflects the maximum memory consumption required to address the GP problem with N vertices. In this context, the number of logical qubits establishes the following rule:

$$l = [\mathbb{1}(k = 2) + k\mathbb{1}(k > 2)] \min\left(k^d, \left\lceil \frac{N}{2k^{d-1}} \right\rceil 2k^{d-1}\right), \tag{3.7}$$

where $d = \lceil \log_k N \rceil$, i.e. the smallest integer greater than $\log_k N$, with N being the number of the qubit lines of the circuit and $k = 2, 3, 4$ being the number of partitions. Intuitively, (3.7) implies that the depth of the GP tree (the largest number of edges from the root node to the most distant leaf node) equals d defined above. Thus, there are $d+1$ subgraph (partition) levels and at each level i (except the last) each subgraph

$G_j^{(i)}$, $1 \leq j \leq k^{i-1}$ is partitioned into k subgraphs in which nodes correspond to logical qubits, $0 \leq i \leq (d-1)$. As N might be not a power of k, some extra qubits are needed in order to perform balanced GP. Thus, the final number of qubits becomes equal either k^d, or double of the greatest power of k smaller than N, i.e. $2k^{d-1}$. Such a mechanism allows us to bound the order of leaves from above by 2. However, to encode the GP-problem into QUBO, we need to construct a matrix with k characteristic vectors as its rows (3.3) for $k = 3,\ 4$. It explains the presence of multiplier k in (3.7) for $k = 3,\ 4$. For the case $k = 2$, there is no need to introduce k characteristic vectors, as binary vector $\mathbf{x} = \{x_i\}_{i=1}^N$ defines which part qubit i belongs to (if $x_i = 0$, then qubit i belongs to the 1st partition, and to the second one, otherwise) as it was implemented in Algorithm 1. Note that extra qubits are omitted in the final permutation of qubits.

Table 3.7: Numerical results of the summary memory requirements presented as a pair of number of logical and physical (in parentheses) qubits required to optimize QCs from Section 3.3.4, and Sections 3.4.1–3.4.4 by QA.

k	Circuit from [24]	double Toffoli	GF(2^4) multiplier	Hamming gate	2-to-4 decoder
2	24 (28)	8 (8)	64 (84)	24 (28)	8 (8)
3	54 (103)	18 (32)	108 (331)	54 (101)	18 (30)
4	32 (90)	16 (28)	128 (433)	32 (95)	16 (28)

In Table 3.7, the numerical estimates for the summary memory requirements displayed as pairs indicating the number of logical and physical (given in parentheses) qubits, as obtained from our experiments. Each count of logical qubits represents the sum of logical qubits at each level of partitioning, approximated by (3.7).

3.5 Discussion

In this chapter, we propose a hybrid QC optimization approach that minimizes NNC and is based on balanced partitioning of the qubit line adjacency graph by QA. Experiments on various QCs performed using D-Wave machine demonstrate applicability of our approach. Intuitively, as we keep gate order within the optimized QC, optimization procedure returns correct (valid) QC. However, there is a software `Quantify` [89] specifically designed for verification of QCs after optimization. Sensitivity to the parameters of the proposed approach was studied, and reasonable parameter values demonstrate good results (optimized circuits). The problem of tuning the penalty constant up, however, implies metaoptimization and should be studied separately.

Note that to consider 2D topology instead of linear one is to replace modulus with the Manhattan distance in (3.1) as this kind of distance is used in performing optimal qubit circuit conversion to 2D NN architecture [114]. However, the function to be optimized will be a matrix function with number of rows and columns equal to the square of the number of qubits, so further research is needed. Interestingly, the GP problem arises quite naturally in the problems of compilation for distributed quantum computers [32] making the use of the quantum annealer quite promising in that context.

However, qubits can also be reordered locally. It implies to rearrange the circuit's gates and combine them into separate NN-subcircuits without changing the circuit logic. For that, qubits are reordered each time for each NN-subcircuit. Such a flexibility of local qubit reordering makes it promising.

To make our thoughts clear, we assume that it is possible to rearrange gates without changing circuit logic. We wonder, if there is such a gate rearrangement that optimizes $SWAP$ count to make the circuit NN-compliant.

Let us consider double Toffoli gate (see Figure 3.9) again. As follows from Figure 3.28, it can be divided into 2 LNN-compliant subcircuits where qubits are reordered as $\begin{bmatrix} q_3 & q_0 & q_1 & q_2 \end{bmatrix}$ in the 1st subcircuit, and $\begin{bmatrix} q_0 & q_3 & q_1 & q_2 \end{bmatrix}$ in the 2nd one. Only 1 $SWAP$ is needed to make these permutations consistent (to swap qubit q_3 and qubit q_0).

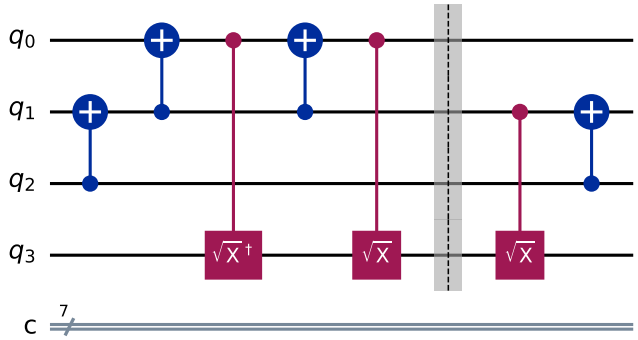


Figure 3.28: Double Toffoli gate divided into 2 LNN-compliant subcircuits. 1 $SWAP$ is needed to make it LNN-compliant.

Recall that global qubit reordering requires one $SWAP-$pair, i.e. 2 $SWAP$s, to make double Toffoli gate LNN-compliant. Thus, local qubit reordering is promising and worth the detailed investigation that is presented in Chapter 4.

Chapter 4

Quantum Circuit Optimization for 2D NN Architecture by solving Boolean Satisfiability Problem using Quantum Annealing

As advertised above, here we focus on optimizing QCs through the local reordering of qubit lines within a 2D NN architecture. Several approaches have been proposed for QC optimization in 2D NN architectures [102, 118, 43, 114]. We build upon the methodology introduced in [114] where the authors consider a quantum circuit as a gate dependency graph and divide it into NN-compliant subcircuits by reordering the qubits locally. To achieve this, they propose to partition the graph using Boolean satisfiability (already been studied in quantum computing [73, 18, 15, 88]). Further, to connect 2D placements of qubits obtained for each subcircuit, A^* heuristic search is used as it generally allows to find optimal path between any two nodes of the graph. The novelty of the present approach is that it is hybrid quantum-quantum QC optimization approach with quantum computing techniques replacing the classical counterpart. The choice of such a revolutionary type of computing is the main advantage of our approach. More specifically, QA implemented in D-Wave quantum machine is proposed to solve the problem under study. Being a hardware implemented heuristic, it returns the solution in constant time, opposed to traditional classical heuristics. It makes QA beneficial for solving QC optimization problem. The details are described further.

Note that this chapter represents a reworked version of the article [79] enriched with supplemental numerical experiments that include the analysis and performance

evaluation of various quantum operations, such as Toffoli gate, Double Toffoli gate, and 2-to-4 decoder.

4.1 Problem Statement

We study possibilities of QC optimization via local qubit line mapping in 2D NN architecture. For such an architecture, gate set is commonly restricted and consists of single-qubit gates and $CNOT$. In combination, the aforementioned gates form a universal gate set meaning that any unitary operation can be approximated by a QC composed of only these gates [87].

To introduce a 2D architecture, a 2D grid is defined and each qubit has four neighbors (two horizontal and two vertical) at most here. For instance, in grid

$$\begin{bmatrix} q_1 & q_2 & q_3 \\ q_4 & q_5 & q_6 \end{bmatrix}$$

qubit q_1 has two neighboring qubits which are q_2 and q_4.

In 2D NN architecture, two-qubit gates act only on adjacent qubits, and $SWAP$ gates are used to rearrange non-adjacent interacting qubit lines to perform given operations. Count of additional $SWAP$ gates depends on the initial qubit lines placement. Therefore, finding a 2D placement of qubit lines that minimizes the $SWAP$ count is essential for optimizing the circuit.

More precisely, we have to find a sequence of qubit line placements $\{A^l\}_{l=1}^m$ meaning that qubit line placement is changed for, say, m times, where A^l is a 2D grid (matrix) with ${A^l}_{ij} = q$ if qubit q is positioned at $(i,\ j)$. For convenience, we also define inverse mapping $q \to ({i^l}_q, {j^l}_q)$ as a position of qubit q in 2D placement A^l. Such a sequence consists of qubit line placements each of which is valid only for one subcircuit constructed from a given circuit by gates recombination and becoming NN-compliant after qubit line reordering. To map each intermediate qubit line placement A^l to the next one, additional $SWAP$ gates are needed. For a 2D grid, its count is calculated via the Manhattan distance and should be minimized, i.e.

$$s(l) := \sum_{q=1}^{n} \left|{i^l}_q - {i^{l+1}}_q\right| + \left|{j^l}_q - {j^{l+1}}_q\right| - 1 \to \min\,. \tag{4.1}$$

Then, the overall $SWAP$ count within the whole circuit is as follows:

$$\sum_{l=1}^{m-1} s(l) = \sum_{l=1}^{m-1} \Big(\sum_{q=1}^{n} \left| {i^l}_q - {i^{l+1}}_q \right| + \left| {j^l}_q - {j^{l+1}}_q \right| - 1 \Big) \to \min, \tag{4.2}$$

where n is the number of qubits.

However, quantum gates composing a given circuit should be recombined according to the gate dependency graph $G = \langle V, E \rangle$, where V is a set of gates of the circuit and E is the set of dependencies of the gates with Boolean weight $e_{g_1,g_2} = 1$, if g_2 follows g_1 and they are not commutative (meaning that interchange of them causes different result of circuit implementation), and $e_{g_1,g_2} = 0$, if g_1 and g_2 are commutative.

As such, the problem is a GP problem with goal function (4.2).

Note that the proposed problem statement is also correct for linear architecture if we replace the Manhattan distance with absolute difference between interacting qubits location in (4.2).

4.2 Solution Approach

We introduce a hybrid quantum-quantum $SWAP$ count optimization of a QC to be implemented in a 2D NN architecture following [114]. The key ingredients of the original approach involves Boolean satisfiability (that checks if there is at least one set of the variables' assignments under which the given Boolean formula evaluates to TRUE [18]) and A^* heuristic graph-traversal path-finding search [46].

Algorithmic description of our approach is presented below. It differs from the [114] by solving SAT-problem in QUBO format using QA (see Step 3.2 – Step 3.5 below).

Step 1 **Require**: gate dependency graph G; set of gates Γ; partition $P = \emptyset$;

Step 2 While Γ not empty:

Step 3 Binary search to construct a subcircuit G_l at current iteration l starting from $l = 1$ (taking into account gate dependency graph G)

Step 3.1 Convert G_l to a CNF CNF_l;

Step 3.2 Convert SAT-problem with CNF_l to a QUBO problem with coefficient matrix Q_l (by `qubovert` Python package)

Step 3.3 Solve QUBO problem defined by Q_l using QA

Step 3.4 Extract the first solution (with the lowest energy) - qubit line placement A^l;

Step 3.5 If CNF_l is TRUE,

Step 3.5.1 Add new subcircuit to partition $P = P \bigsqcup G_l$, and exclude gates used from available ones $\Gamma = \Gamma / \ G_l$

Step 3.5.2 Return to **Step 2**;

Step 3.6 Else – return to **Step 3**;

Step 4 Apply A^* search to connect $\{A^l\}$ and calculate $SWAP$ count needed for that.

The approach is iterative and is based on binary search technique used at **Step 3** to construct a subcircuit taking into account gate dependencies. To construct CNF at Step 3.1, the following conditions should be formulated:

Condition 1 Interacting qubits of all gates are adjacent.

Condition 2 Each qubit is assigned to only one cell on a 2D grid.

Condition 3 At most one qubit is assigned to each cell on a 2D grid.

We introduce Boolean variables x_{ijk} that are 1 if qubit q_k is assigned to cell (i, j) and 0 otherwise. For simplicity, 2D grid of 3 rows and 3 columns where cells are placed rowly is considered.

To express Condition 1 as a Boolean function, we consider a CNOT(q_1, q_2) as an example. If q_1 is assigned to (1,1), then q_2 should be assigned to either one of (0,1), (1,0), (1,2), or (2,1). This condition can be expressed as

$$(\neg x_{111}) \vee (x_{012} \vee x_{102} \vee x_{122} \vee x_{212}). \tag{4.3}$$

Then, applying disjunction to the Boolean formulas of such conditions for the other cells of the grid, we obtain a formula for the condition such that q_1 and q_2 should be adjacent on the grid. After that, we apply conjunction of the obtained Boolean formulas for all the pairs of interacting qubits.

As for Condition 2, each qubit is assigned to at least one cell and at most one cell. The former condition can be expressed as

$$\vee_{i,j} x_{ijk} = 1 \tag{4.4}$$

for each qubit q_k. The latter condition prohibits assigning the qubit to 2 different cells and is as follows:

$$\neg x_{ijk} \vee \neg x_{lmk} = 1 \tag{4.5}$$

for each qubit q_k and each pair of cells $(i, j) \neq (l, m)$.

Condition 3 prohibits assigning two different qubits to one cell meaning that

$$\neg x_{ijk} \vee \neg x_{ijl} = 1 \tag{4.6}$$

for each cell (i, j) and each pair of qubits q_k and q_l, $k \neq l$.

At Step 3.3, QA solves QUBO problem constructed from CNF by `qubovert` automatically. This package is used for formulating, simulating, and solving problems in Boolean and spin form; for further details one may refer to [4]. If the solution is satisfiable, i.e. there is such a qubit placement A^l satisfied to Conditions 1, 2, and 3, then G_l forms a NN-compliant subcircuit with qubits placement A^l and binary search is continued on the rest of circuit.

After partitioning the gate dependency graph G into NN-compliant subcircuits with qubit placements $\{A^l\}_{l=1}^m$, A^* search is used to connect them optimally (**Step 4**). A^* search is a heuristic that allows to find optimal way from the starting node S of a graph to the target one T based on a cost function $f(u) = g(u) + h(u)$, where u is a current node during the search. $g(u)$ is a cumulative cost from S to u [114] and $h(u)$ is a heuristic that estimates the cost from the current qubit placement u to T. In our problem, a qubit placement A^l is a node of a graph with A^1 as the starting node and A^m as the target one. Cumulative cost $g(A^l)$ is $SWAP$ count needed to obtain A^l from A^1 and it is the sum of the Manhattan distances between the positions of each qubit in the A^1 and A^l, i.e.

$$\sum_{i=1}^{l-1} s(i),$$

where $s(i)$ is defined in (4.1). Heuristic cost $h(A^l)$ is calculated analogously.

4.3 Numerical Experiments

Recall that in order to solve a SAT problem in QA, at first it should be converted into QUBO format (2.6). It can be done using the technique from [18] which requires preliminary setup to encode a SAT-problem to Ising model. Alternative way is to use Python package `qubovert` that performs such a conversion automatically by using `sat`

library. Here, we propose to follow the second approach. To obtain a SAT-problem, each of the Conditions 1, 2, and 3 are added to the resulting Boolean formula with some positive Lagrangian parameter tuned by hand. For the experiments performed, $\lambda = 100$ for (4.3), (4.5), and (4.6), and $\lambda = 10$ for (4.4).

As the proposed approach is based on taking into account gate dependencies, we use `CommutationAnalysis` package provided by `qiskit.transpilers.passes` to establish correct gate dependencies. Although, there is an appropriate software `Quantify` [89] allowing one to perform verification of the optimized QCs.

To verify the applicability of the approach, we use several quantum gates from `Reversible logic synthesis benchmark page by D. Maslov` (available at `https://reversiblebenchmarks.github.io/`) and from `RevLib` benchmark [117].

All the experiments performed by using SA and classical SAT-solver `PicoSAT` are run on local Ubuntu machine (Ubuntu 23.04 with 31 GiB RAM using 11th Gen Intel(R) Core(TM) i7-11700 @ 2.50GHz). To perform the experiments using QA, we use a real quantum annealer, namely, `D-Wave Advantage system 5.4` with 5614 qubits and 40050 couplers, fixing the annealing time to $20\mu s$ and taking the best out of 2000 reads. The software of the proposed approach is available via link `https://github.com/MakarMariia97/Quantum-Annealing-for-Local-Qubit-Reordering`.

Note that numerical experiments are performed only for the circuits with no more than 5 qubits because of the high computational demand on `qubovert`. Along the lines, we also benchmark the resulting quantum part of the algorithm in terms of logical and physical qubits to demonstrate the computational complexity of QA.

4.3.1 Fredkin Gate Optimization

Fredkin gate is a gate from `Reversible logic synthesis benchmark page by D. Maslov`. It is frequently used in quantum computing because of its small cost in some quantum computing technologies.

The circuit consists of 7 quantum gates (enumerated as $C_1, \ldots, C_7$) operating on different pairs of 3 qubits and is illustrated in Figure 4.1. In case when qubits are placed in 2×2 grid rowly with numbering starting at the left side, i.e. NN topology defined as $A = \begin{bmatrix} q_0 & q_1 \\ q_2 & - \end{bmatrix}$, 6 $SWAP$s are used to implement the circuit. To explain it, consider each gate included in the circuit sequentially. The first gate operates on q_2 and q_1, that are non-adjacent on the grid. Thus, by swapping q_1 and q_0, we make q_2 and q_1 adjacent. After this gate, we need to restore initial qubit placement by swapping

this pair of qubits again; thus, we use two $SWAP$s for the first gate. The second gate operates on q_0 and q_2 that are adjacent on the grid. Like the first gate, the third gate also requires two $SWAP$s. The fourth gate operates on q_0 and q_1 that are adjacent; no auxiliary $SWAP$s are used for this gate. The fifth and sixth gates work on q_1 and q_2, as the first gate does; thus, they require two $SWAP$s (one – before the fifth gate, and one – after the sixth one). The last gate works on qubits that are already adjacent. In total, we use 6 $SWAP$s to make Fredkin gate NN-compliant on 2D grid straightforwardly. In Figure 4.2, the dependency of the gates of Fredkin gate is illustrated.

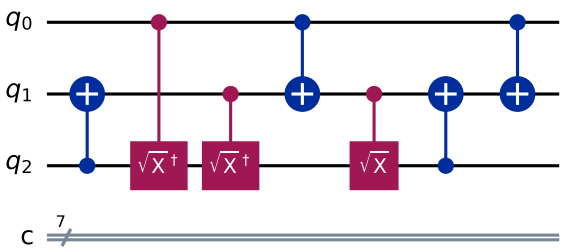


Figure 4.1: Fredkin gate. Initial $SWAP$ count is 6.

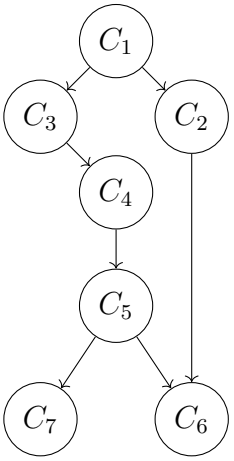


Figure 4.2: Dependency of gates in Fredkin gate from Figure 4.1.

Simulated Annealing

At first, we use SA due to the limited access time of QA on D-Wave. After applying the proposed approach, the circuit is divided into 2 NN-compliant subcircuits as follows from Figure 4.3. The qubits are placed following $A^1 = \begin{bmatrix} q_2 & q_1 \\ q_0 & - \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_1 & q_2 \\ q_0 & - \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_1 and q_2. Thus, 1 $SWAP$ is needed to make these placements consistent.

Quantum Annealing

QA divides the circuit into subcircuits in the same way as SA does. The qubits are placed following $A^1 = \begin{bmatrix} - & q_0 \\ q_1 & q_2 \end{bmatrix}$ and $A^2 = \begin{bmatrix} - & q_0 \\ q_2 & q_1 \end{bmatrix}$ within the 1st and the 2nd sub-

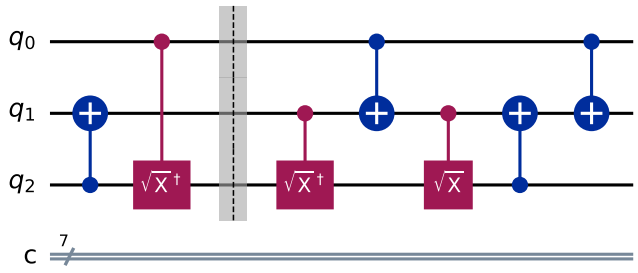


Figure 4.3: The optimized Fredkin gate from Figure 4.1.

circuits, respectively. Similar to SA, 1 $SWAP$ is used to transit from A^1 to A^2 (by swapping q_1 and q_2).

In order to convert the subcircuit to QUBO, 19 logical qubits and 36 physical qubits are used for the 1st subcircuit whereas the 2nd subcircuit holds 33 physical qubits. The dimensions of the constructed QUBO problems are appropriate for the current experiments, which utilize D-Wave's client operating on the `Advantage system 5.4` chip, featuring 5614 qubits.

Classical SAT-solver

In order to validate the result obtained above, we replicate the classical approach proposed in [114] and use the classical SAT solver `PicoSAT` via `pyeda` Python package. In this experiment, the circuit is divided in the same way as both in QA and SA (see Figure 4.3). The qubits are placed following $A^1 = \begin{bmatrix} - & q_1 \\ q_0 & q_2 \end{bmatrix}$ and $A^2 = \begin{bmatrix} - & q_2 \\ q_0 & q_1 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. 1 $SWAP$ is used to transit from A^1 to A^2 (by swapping q_1 and q_2).

4.3.2 CNOT-based Circuit Optimization

To verify the applicability of the proposed approach for a bigger QC, we construct a circuit illustrated in Figure 4.4 containing 6 CNOT gates (enumerated as $C_1, \ldots, C_6$) operating on some pairs of 5 qubits and try to minimize $SWAP$ gate count needed to implement the circuit in 2D NN architecture with 2 rows and 3 columns. Initially, it requires 6 $SWAP$s for 2D NN architecture where qubits are placed rowly, i.e. $A = \begin{bmatrix} q_0 & q_1 & q_2 \\ q_3 & q_4 & - \end{bmatrix}$. To explain it, consider each gate included in the circuit sequentially. The

first gate operates on q_2 and q_0, that are non-adjacent on the grid. Thus, by swapping q_1 and q_2, we make q_2 and q_0 adjacent. After this gate, we need to restore initial qubit placement by swapping this pair of qubits again; thus, we use two $SWAP$s for the first gate. The second gate operates on q_1 and q_3 that are non-adjacent on the grid. By swapping q_0 and q_1 before the gate, we make them adjacent; and we swap them back to restore initial placement. The third, fourth, and fifth gates already work on adjacent qubits. Like the first gate, the last one also requires two $SWAP$s. In total, we use 6 $SWAP$s to make the QC NN-compliant on 2D grid straightforwardly. In Figure 4.5, the dependency of the gates of circuit from Figure 4.4 is illustrated.

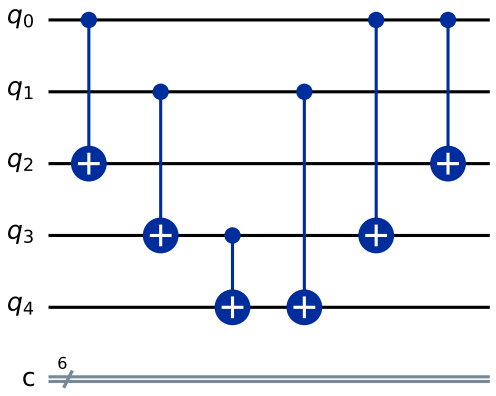


Figure 4.4: The circuit to be optimized.

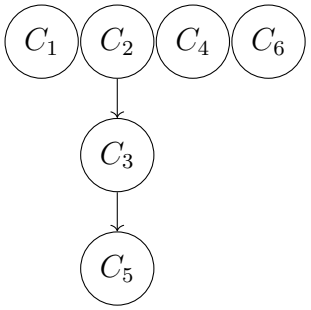


Figure 4.5: Dependency of gates in circuit from Figure 4.4.

Simulated Annealing

After applying SA, the number of $SWAP$s needed is 2, at best. During the optimization, the circuit is divided into 2 NN-compliant subcircuits: the first one contains C_1, C_2, C_3, C_5, and C_6, and C_4 gate forms the second subcircuit (see Fig 4.6). Note that C_4 and C_6 gates are commutative, and C_4 can be placed after C_6. The qubits are placed following $A^1 = \begin{bmatrix} - & q_4 & q_2 \\ q_1 & q_3 & q_0 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_3 & - & q_2 \\ q_1 & q_4 & q_0 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_3 and q_4 and then transit q_3 to the top-left.

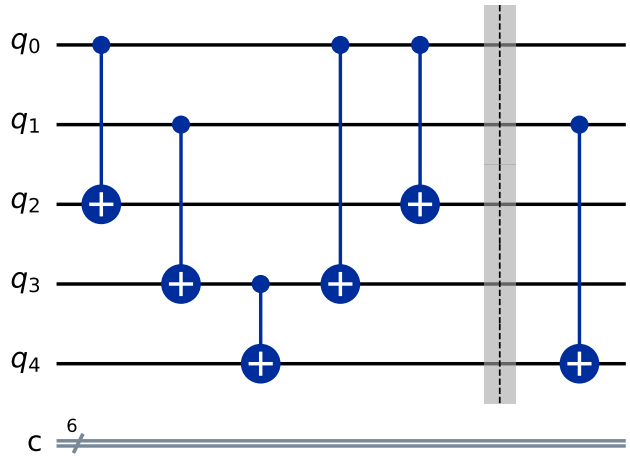


Figure 4.6: The optimized circuit from Figure 4.4.

Quantum Annealing

QA also divides the circuit from Figure 4.4 in the same way as SA does (see Figure 4.6) where optimal qubits placements follows $A^1 = \begin{bmatrix} q_4 & q_3 & q_0 \\ - & q_1 & q_2 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_0 & q_1 & q_4 \\ - & q_2 & q_3 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. We should note that SA always returns positive result in case when a subcircuit is satisfiable unlike QA. Thus, we should repeat QA for several times (it is 15 in our case) to reduce the probability to exclude satisfiable subcircuit accidentally. In order to make these optimal qubit line placements consistent, 5 $SWAP$s are used. At first, we swap q_3 and q_4, following swapping q_0 and q_4 that results in $\begin{bmatrix} q_3 & q_0 & q_4 \\ - & q_1 & q_2 \end{bmatrix}$. After that, we swap q_0 and q_3, and then – q_1 and q_3 that results in $\begin{bmatrix} q_0 & q_1 & q_4 \\ - & q_3 & q_2 \end{bmatrix}$. Finally we swap q_2 and q_3, and obtain A^2.

To encode the first subcircuit into QUBO, 103 logical and 354 physical qubits are used, and 111 logical and 304 physical qubits are used for the encoding of the second subcircuit into QUBO. The dimension of the constructed QUBO problems is suitable as the current experiments are performed using D-Wave's client based on `Advantage system 5.4` chip with 5614 qubits.

Here, QA returns suboptimal solution due to the dimension of the problem. As this CNOT-based circuit requires more logical and physical qubits than Fredkin gate investigated in Section 4.3.1, it seems that more simulation runs are needed to obtain better solution. However, its number is bounded by the access time provided by D-Wave

machine that is also limited. We decided to increase the number of simulation runs up to 4000 and the result is encouraging. The proposed technique partitions the circuit in the same way with qubit placements $A^1 = \begin{bmatrix} q_2 & q_0 & - \\ q_4 & q_3 & q_1 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_0 & - & q_4 \\ q_2 & q_3 & q_1 \end{bmatrix}$. Only 3 $SWAP$s are needed to make the subcircuits consistent.

Classical SAT-solver

By optimizing the circuit using `PicoSAT`, the circuit is divided in the same way as both in QA and SA (see Figure 4.6). Optimal qubits placements of the subcircuits follows $A^1 = \begin{bmatrix} - & q_4 & q_2 \\ q_1 & q_3 & q_0 \end{bmatrix}$ and $A^2 = \begin{bmatrix} - & q_4 & q_0 \\ q_3 & q_1 & q_2 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_0 with q_2, and q_1 with q_3. Thus, 2 $SWAP$s are used to make these placements consistent.

4.3.3 Toffoli Gate Optimization

Toffoli gate is a universal gate from `Revlib`. It consists of 2 control and 1 target qubits. In QC hardware, it cannot be implemented in its initial way and that's why it is decomposed as follows from Figure 4.7. If qubits are placed like $A = \begin{bmatrix} q_0 & q_1 \\ q_2 & - \end{bmatrix}$ in 2D then 4 $SWAP$s are used to implement the circuit in 2D NN architecture. To explain it, consider each gate included in the circuit sequentially. The first second gates operates on q_2 and q_0, that are adjacent on the grid. The second and fourth gates act on adjacent qubits q_0 and q_1. The third and fifth gates operate on qubits q_1 and q_2, that that become adjacent on the grid after swapping q_0 and q_1. After each of these gates we swap this pair of qubits back to restore initial placement. In total, we use 4 $SWAP$s to make Toffoli gate NN-compliant on 2D grid straightforwardly. In Figure 4.8, the dependency of the gates of circuit from Figure 4.7 is illustrated.

Simulated Annealing

At first, we use Simulated Annealing due to the limited access time of QA on D-Wave. After applying the proposed approach, 1 $SWAP$ is needed at best. During the optimization, the circuit was divided into 2 NN-compliant subcircuits: the first one contains the first 2 gates and the other gates form the second subcircuit (see Fig 4.9). The qubits are placed following $A^1 = \begin{bmatrix} q_2 & q_0 \\ - & q_1 \end{bmatrix}$ and $A^2 = \begin{bmatrix} - & q_0 \\ q_2 & q_1 \end{bmatrix}$ within the 1st and

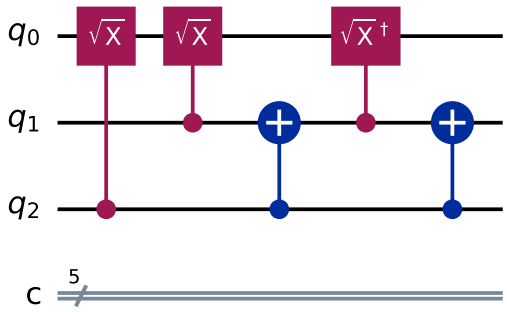


Figure 4.7: Toffoli gate to be optimized.

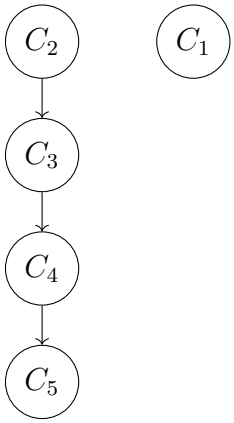


Figure 4.8: Dependency of gates in implementation of Toffoli gate from Figure 4.7.

the 2nd subcircuits, respectively. To transit from A^1 to A^2, we place q_2 in bottom-left.

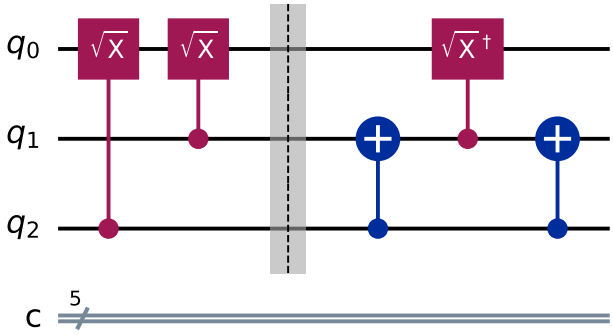


Figure 4.9: The optimized Toffoli gate implementation from Figure 4.7.

Quantum Annealing

QA also divides the circuit from Figure 4.7 in the same way as SA does (see Figure 4.9) where the optimal qubit placements for the subcircuits are as follows: $A^1 = \begin{bmatrix} - & q_2 \\ q_1 & q_0 \end{bmatrix}$ and $A^2 = \begin{bmatrix} - & q_2 \\ q_0 & q_1 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_0 and q_1. Thus, only 1 $SWAP$ is used.

In order to convert the 1st subcircuit to QUBO, 20 logical qubits and 37 physical qubits are used, and 19 logical qubits and 34 physical qubits are used for the 2nd

subcircuit. The dimensions of the constructed QUBO problems are appropriate for the current experiments, which utilize D-Wave's client operating on the `Advantage system 5.4` chip, featuring 5614 qubits.

Classical SAT-solver

In order to validate the result obtained above, we use the classical SAT solver `PicoSAT` via `pyeda` Python package. In this experiment, the circuit is divided in the same way as both in QA and SA (see Figure 4.9). The qubits are placed following $A^1 = \begin{bmatrix} - & q_2 \\ q_1 & q_0 \end{bmatrix}$ and $A^2 = \begin{bmatrix} - & q_2 \\ q_0 & q_1 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transform from A^1 to A^2, we swap q_0 and q_1. Thus, 1 $SWAP$ is required to make these placements consistent.

4.3.4 2-to-4 Decoder Gate Optimization

Another circuit to optimize is 2-to-4 binary decoder from `Revlib` benchmark. This circuit decodes information from 2 inputs into a 4-bit code. The circuit is illustrated in Figure 4.10. In case if qubits are placed rowly in 2×2 grid $A = \begin{bmatrix} q_0 & q_1 \\ q_2 & q_3 \end{bmatrix}$, 6 $SWAP$s are used to implement the circuit. To explain it, consider each gate included in the circuit sequentially. The first gate operates on q_2 and q_1, that are non-adjacent on the grid. Thus, by swapping q_1 and q_0, we make q_2 and q_1 adjacent. After this gate, we need to restore initial qubit placement by swapping this pair of qubits again; thus, we use two $SWAP$s for the first gate. The second gate operates on q_1 and q_3 that are adjacent on the grid. The third gate operates on q_3 and q_0, that become adjacent after swapping q_0 and q_1. One $SWAP$ is used after this gate to restore initial qubit placement. The fourth gate already works on adjacent qubits. Like the first gate, the fifth and sixth gates act on q_1 and q_2. As the sixth gate follows the fifth one immediately, we use one $SWAP$ before the fifth gate, and one $SWAP$ after the sixth one. The last two gates operate on qubits that are adjacent on the grid; no auxiliary $SWAP$s are used. In total, we use 6 $SWAP$s to make 2-to-4 decoder gate NN-compliant on 2D grid straightforwardly. In Figure 4.11, the dependency of the gates of circuit from Figure 4.10 is illustrated.

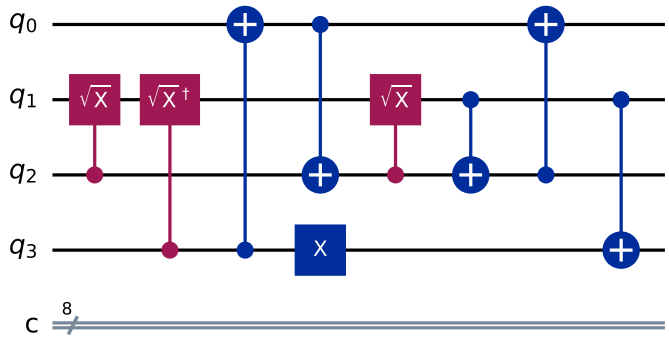


Figure 4.10: 2-to-4 binary decoder to be optimized.

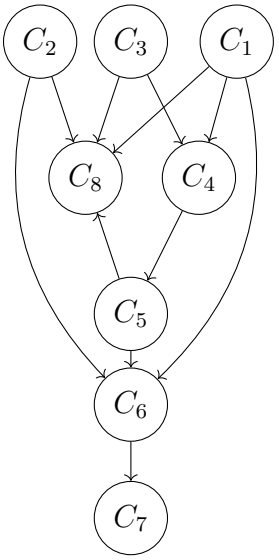


Figure 4.11: Dependency of gates in circuit from Figure 4.10.

Simulated Annealing

At first, we use Simulated Annealing due to the limited access time of QA on D-Wave. After applying the proposed approach, no $SWAP$s are required because the circuit becomes NN-compliant if qubits are placed as follows $A = \begin{bmatrix} q_3 & q_1 \\ q_0 & q_2 \end{bmatrix}$.

Quantum Annealing

QA also finds that the circuit becomes NN-compliant if qubits are placed in the following way: $A = \begin{bmatrix} q_2 & q_0 \\ q_1 & q_3 \end{bmatrix}$. In order to convert the problem to QUBO, 19 logical qubits and 66 physical qubits are used.

Classical SAT-solver

In order to validate the result obtained above, we use the classical SAT solver `PicoSAT` via `pyeda` Python package. In this experiment, the circuit also becomes NN-compliant after the qubits reordering following $A = \begin{bmatrix} q_3 & q_0 \\ q_1 & q_2 \end{bmatrix}$.

4.3.5 Double Toffoli Gate Optimization

Another circuit to optimize is the double Toffoli gate presented in `Revlib` benchmark. The circuit is illustrated in Figure 4.12. In case if qubits are placed rowly in 2×2 grid $A = \begin{bmatrix} q_0 & q_1 \\ q_2 & q_3 \end{bmatrix}$, 8 $SWAP$s are used to implement the circuit. To explain it, consider each gate included in the circuit sequentially. The first gate operates on q_2 and q_1, that are non-adjacent on the grid. Thus, by swapping q_1 and q_0, we make q_2 and q_1 adjacent. After this gate, we need to restore initial qubit placement by swapping this pair of qubits again; thus, we use two $SWAP$s for the first gate. The second gate operates on q_1 and q_0 that are adjacent on the grid. The third and fifth gates operate on q_3 and q_0, that become adjacent after swapping q_0 and q_1. One $SWAP$ is used after each of these gates to restore initial qubit placement. The fourth and sixth gates already work on adjacent qubits. Like the first gate, the last one acts on q_1 and q_2, and requires two $SWAP$s. In total, we use 8 $SWAP$s to make 2-to-4 decoder gate NN-compliant on 2D grid straightforwardly. In Figure 4.13, the dependency of the gates of circuit from Figure 4.12 is illustrated.

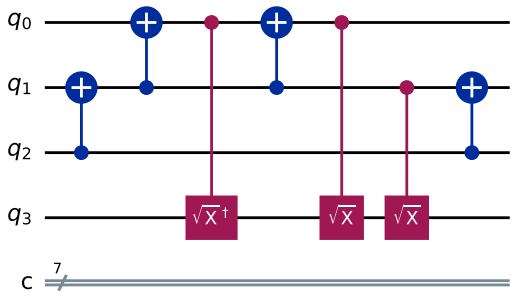


Figure 4.12: Double Toffoli gate.

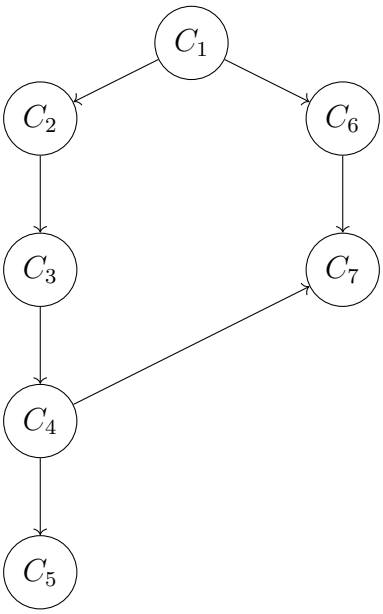


Figure 4.13: Dependency of gates in circuit from Figure 4.12.

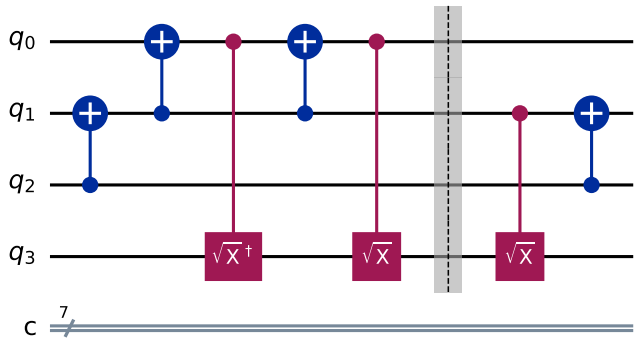


Figure 4.14: The optimized double Toffoli gate from Figure 4.12.

Simulated Annealing

At first, we use Simulated Annealing due to the limited access time of QA on D-Wave. After applying the proposed approach, 2 $SWAP$s are needed at best. During the optimization, the circuit is divided into 2 NN-compliant subcircuits: the first one contains the first 2 gates and the other gates form the second subcircuit (see Fig 4.14). The qubits are placed following $A^1 = \begin{bmatrix} q_1 & q_2 \\ q_0 & q_3 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_1 & q_3 \\ q_2 & q_0 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_2 and q_3 firstly, and then swap q_0 and q_2.

Quantum Annealing

QA also divides the circuit from Figure 4.12 in the same way as SA (see Figure 4.14) where the optimal qubit placements for the subcircuits are as follows: $A^1 = \begin{bmatrix} q_1 & q_0 \\ q_2 & q_3 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_3 & q_0 \\ q_1 & q_2 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_2 and q_3 firstly, and then swap q_1 and q_3. Thus, 2 $SWAP$s are used to make these placements consistent.

In order to convert the 1st subcircuit to QUBO, 26 logical qubits and 54 physical qubits are used, and 26 logical qubits and 50 physical qubits are used for the 2nd subcircuit. The dimensions of the constructed QUBO problems are appropriate for the current experiments, which utilize D-Wave's client operating on the `Advantage system 5.4` chip, featuring 5614 qubits.

Classical SAT-solver

In order to validate the result obtained above, we follow the approach proposed in [114] and use the classical SAT solver `PicoSAT` via `pyeda` Python package. In this experiment, the circuit is divided in the same way as both in QA and SA (see Figure 4.14). The qubits are placed following $A^1 = \begin{bmatrix} q_3 & q_0 \\ q_2 & q_1 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_3 & q_0 \\ q_1 & q_2 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. To transit from A^1 to A^2, we swap q_1 and q_2. Thus, 1 $SWAP$ is required to make these placements consistent.

4.4 Discussion

In this chapter, we introduce hybrid QC optimization approach by local reordering of qubits using QA and Boolean satisfiability. Its promising nature is explained by the use of quantum computing for solving such a problem. Numerical experiments show that QA performs well on small scale: the solution obtained is comparable with one returned by classical SAT-solver. However, while solving a larger problem, QA may return suboptimal results, and making the simulation process longer improves the solution.

Besides that, there is a modified version of QA [90] that incorporates a tabu mechanism to penalize suboptimal solutions. We propose to explore it and apply to the problem of QC optimization. Specifically, we suggest integrating this approach into the optimization process through the local qubit reordering and verifying its effectiveness by conducting numerical experiments. The following chapter will present the main results, along with the findings from the numerical experiments.

Chapter 5

Tabu Enhanced Quantum Annealing based Quantum Circuit Optimization

Nowadays, various quantum-classical hybrid schemes are introduced to widen the class of problems that can be solved with the help of quantum computing and they use it as one of the stages in the algorithm, e.g. by means of the so-called variational hybrid quantum-classical optimization [49] or, more widely, variational quantum algorithms (on a generic quantum hardware) [40], stochastic gradient descent methods [110] or hybrid tabu search [71], to name a few. In the latter case, quantum effects are used to overcome the traps in local optima, while tabu memory improves forecasting accuracy [71]. In the same direction of memory-based improvement of the quantum computing algorithm, a recent paper [90] proposed a hybrid quantum-classical algorithm based on QA to solve arbitrary QUBO problems, named Quantum Annealing Learning Search (QALS). A further development of the idea of learning search was presented in [91] in the context of a generic optimization problem solved by means of adiabatic quantum computing.

Convergence of the algorithm is a necessary step in the analysis of any heuristics. For a non-deterministic algorithm, the usual way is to construct the corresponding stochastic process (possibly a Markov process) and observe its convergence to a steady state. In such a way, convergence of QA was established by considering inhomogeneous Markov chain in [85], whereas various stochastic processes related notions such as hitting times and weak convergence were used in the case of the so-called quantum walks [9, 98, 77, 12]. However, in general memory comprises the theoretical convergence due to a (theoretically unlimited) dependence on the previous states of the algorithm's trajectory. Thus, it is important to reason the finiteness of the memory of corresponding tabu search mechanism [42, 50, 55, 47].

In this chapter, we investigate the theoretical convergence of QALS by introducing collisions in the object that stores the tabu states, based on the Ising model. We refer to the modified algorithm as Tabu-Enhanced Quantum Annealing Hybrid Quantum Optimization (TEQAHQO). Additionally, we propose applying this approach to quantum circuit optimization through the local qubit reordering, as introduced in Chapter 4.

This chapter is a reworked version of the article [19] which introduces a more general technique (called TEHQO) that uses both QA and AQC to solve optimization problems. Additionally, we expand upon the article by applying TEQAHQO to the QC optimization problem studied and present corresponding numerical experiments.

5.1 Tabu Enhanced Quantum Annealing based Hybrid Quantum Optimization

In this section, we introduce the TEQAHQO approach to solving optimization problems using QA. The QA generalities are presented in Section 2.2.3. For necessary details on QA, as well as QC in general, the reader is referred to e.g. [87, 57]. Here, we recall the basics of QA procedure.

QA is a physical process that can be used as a heuristic search to solve optimization problems [64]. The QA procedure is implemented by a time evolution (the *cooling*) of the quantum system, characterized by the energy dissipation and decoherence, towards the ground state of the problem Hamiltonian H_P. The time evolution follows

$$H(t) = (1 - s(t))H_I + s(t)H_P, \;\; t \in [0, \tau] \tag{5.1}$$

where $s : [0, \tau] \to [0, 1]$ is a smooth monotone function such that $s(0) = 0, \;\; s(\tau) = 1$. QA is closely related to adiabatic quantum computing (described in Section 2.2.2) since the solution of the problem is encoded into the ground state of a problem Hamiltonian, however the considered quantum system is not isolated and the evolution is non-unitary so QA does not simulate universal QC in general.

There is a conceptual analogy between QA and classical SA. The latter is a local search optimization heuristic, a representative of the so-called Generalized Hill Climbing class of algorithms [63], that resembles the annealing physical process in the metal production. The remarkable difference between QA and SA is that QA exploits the *tunnel effect*, a purely quantum phenomenon, to escape local minima instead of thermal hill-climbing, which, in turn, makes QA indeed a *global* search.

Algorithm 5 TEQAHQO scheme

Require: Function $f : \mathcal{Z} \to \mathbb{R}$ to be minimized on $\mathcal{Z} = \{-1, 1\}^n$
Require: maximum temperature T_{max}, function h for temperature decreasing depending on temperature decreasing rate $\eta > 0$
Require: parametric Hamiltonian $H^{(\boldsymbol{\alpha})}$, randomized initialization function ζ and randomized temperature-dependent modification function ξ_T for the parameters $\boldsymbol{\alpha}$, both functions depending explicitly on f
Require: number N of iterations with constant temperature
Require: probability q of performing ground state measurement
Require: maximum number of iterations N_{max} and i_{max}
Ensure: $\boldsymbol{z}^\star \in \mathcal{Z}$ vector minimum of f

$T \leftarrow T_{max}, \boldsymbol{S} \leftarrow \boldsymbol{O}$
initialize $\boldsymbol{\alpha}_1 \leftarrow \zeta(f)$ and $\boldsymbol{\alpha}_2 \leftarrow \zeta(f)$ ▷ Initialize parameters of the Hamiltonian
initialize Hamiltonians $H_P^{(\boldsymbol{\alpha}_1)}(\boldsymbol{O})$ and $H_P^{(\boldsymbol{\alpha}_2)}(\boldsymbol{O})$ as in (5.8)
find $\boldsymbol{z}_1$ and $\boldsymbol{z}_2$ by QA procedure
evaluate $f(\boldsymbol{z}_1)$ and $f(\boldsymbol{z}_2)$
if $f(\boldsymbol{z}_1) \neq f(\boldsymbol{z}_2)$ **then**
    use the best to initialize $\boldsymbol{z}^\star$ and the corresponding parameters $\boldsymbol{\alpha}^\star$.
    use the worst to initialize $\boldsymbol{z}'$
    initialize the tabu matrix: $\boldsymbol{S} \leftarrow \boldsymbol{m}(\boldsymbol{z}')$ ▷ Initialize tabu-matrix with penalty to the worst solution z'
end if
$e \leftarrow 0, d \leftarrow 0, i \leftarrow 0$
repeat
    if N divides i **then** ▷ Temperature decrease after N iterations
        $T \leftarrow h(\eta, T)$
    end if
    with probability q set $\boldsymbol{\alpha} \leftarrow \xi_T(\boldsymbol{\alpha}^\star, f)$, initialize the Hamiltonian $H_P^{(\boldsymbol{\alpha})}(\boldsymbol{S})$ using (5.8) and find $\boldsymbol{z}'$ by QA procedure; otherwise sample $\boldsymbol{z}'$ using quantum random number generator
    if $\boldsymbol{z}' \neq \boldsymbol{z}^\star$ **then**
        evaluate $f(\boldsymbol{z}')$
        if $f(\boldsymbol{z}') < f(\boldsymbol{z}^\star)$ **then** ▷ New candidate solution
            $swap(\boldsymbol{z}', \boldsymbol{z}^\star)$, $\boldsymbol{\alpha}^\star \leftarrow \boldsymbol{\alpha}$, $e \leftarrow 0, d \leftarrow 0$
        else ▷ Suboptimal acceptance
            $d \leftarrow d + 1$ ▷ Counts the number of times that current solution and new candidate solution differ and the former is not worse.
            with probability $e^{-\frac{f(\boldsymbol{z}') - f(\boldsymbol{z}^\star)}{T}}$ $swap(\boldsymbol{z}', \boldsymbol{z}^\star)$, $\boldsymbol{\alpha}^\star \leftarrow \boldsymbol{\alpha}$, $e \leftarrow 0$
        end if
        $\boldsymbol{S} \leftarrow \boldsymbol{S} + \boldsymbol{m}(\boldsymbol{z}')$ ▷ Tabu matrix update
    else
        $e \leftarrow e + 1$ ▷ Counts the number of consecutive times that the current solution is generated.
    end if
    $i \leftarrow i + 1$
until $i = i_{max}$ or $d + e > N_{max}$
return $\boldsymbol{z}^\star$

QA can be physically realized considering a quantum spin glass that is a network of qubits arranged on the vertices of a graph $\langle V, E\rangle$, with $|V| = n$, whose edges in E represent the couplings among the qubits. The total energy of such a system is represented by the Hamiltonian $H_P \equiv H_{\text{Ising}}(\boldsymbol{\Theta})$ of the quantum spin glass,

$$H_{\text{Ising}}(\boldsymbol{\Theta}) = \sum_{i\in V} \theta_i \sigma_3^{(i)} + \sum_{(i,j)\in E} \theta_{ij} \sigma_3^{(i)} \sigma_3^{(j)}. \tag{5.2}$$

$H_{\text{Ising}}(\boldsymbol{\Theta})$ is an operator on the n-qubit Hilbert space $\mathsf{H} = (\mathbb{C}^2)^{\otimes n}$ where the i-th qubit term is represented by a tensor product,

$$\sigma_3^{(i)} := \boldsymbol{I}_{2^{i-1}} \otimes \sigma_3 \otimes \boldsymbol{I}_{2^{n-i}}, \quad i = 1, \ldots, n,$$

where $\boldsymbol{I}_k$ is the identity matrix of size k and σ_3 is the so-called Pauli-Z matrix placed in the i-th tensor factor:

$$\sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix},$$

whereas the interaction of i-th and j-th qubits is represented by the term

$$\sigma_3^{(i)} \sigma_3^{(j)} = \boldsymbol{I}_{2^{i-1}} \otimes \sigma_3 \otimes \boldsymbol{I}_{2^{j-i-1}} \otimes \sigma_3 \otimes \boldsymbol{I}_{2^{n-j}}, \quad j > i = 1, \ldots, n.$$

The coefficient matrix $\boldsymbol{\Theta}$ is the symmetric square size-n matrix of the real coefficients of the Hamiltonian H_{Ising}, called *weights*, defined as

$$\boldsymbol{\Theta} = ||\theta_{ij}||_{(i,j)\in V\times V} := \begin{cases} \theta_i, & i = j, \\ \theta_{ij}, & (i,j) \in E, \\ 0, & (i,j) \notin E. \end{cases} \tag{5.3}$$

These coefficients physically correspond to the coupling terms between the qubits, θ_{ij}, and the so-called local fields, θ_i, on the corresponding vertices.

The matrix σ_3 has two eigenvalues corresponding to the binary states of the qubit, $\{-1, 1\}$, and thus, the system (5.2) has the spectrum of eigenvalues corresponding to all possible values of the cost function given by the energy of the well-known *Ising model*:

$$\mathsf{E}(\boldsymbol{\Theta}, \boldsymbol{z}) = \sum_{i\in V} \theta_i z_i + \sum_{(i,j)\in E} \theta_{ij} z_i z_j, \ \boldsymbol{z} = (z_i)_{i\in V} \in \mathcal{Z} = \{-1, 1\}^n.$$

Thus, the annealing procedure takes the system to the ground state whose corresponding

spin configuration is

$$\boldsymbol{z}^\star = \arg\min_{\boldsymbol{z}\in\mathcal{Z}} \mathsf{E}(\boldsymbol{\Theta}, \boldsymbol{z}), \tag{5.4}$$

where $\boldsymbol{z}^\star$ is the optimal solution of the optimization problem encoded into the parameters of the Hamiltonian $H_{\text{Ising}}(\boldsymbol{\Theta})$ defined in (5.2).

Given a problem, the QA is initialized by a suitable choice of the weights $\boldsymbol{\Theta}$ and the binary variables $z_i \in \{-1, 1\}$ are physically realized by the outcomes of measurements on the qubits located in the vertices V. Thus, since $\boldsymbol{\Theta}$ is symmetric, the optimization problem is specified by $n(n-1)/2$ real parameters satisfying some architectural constraints of a quantum machine,

$$\theta_{ij} \in [-\Delta, +\Delta],$$

for all $(i, j) \in E$ and some finite positive Δ, say, 1. In order to solve a general optimization problem by QA, one needs to obtain the correct *encoding* of the objective function in terms of the cost function $\mathsf{E}(\boldsymbol{\Theta}, \boldsymbol{z})$ which is hard in general.

Now we present the TEQAHQO scheme, a guided meta-heuristic approach designed specifically to solve optimization problems using QA without the need of representing a problem into the quantum architecture a priori.

The key idea of the TEQAHQO is to enhance the QA by adding the so-called *tabu matrix* $\boldsymbol{S}$ to the weight matrix $\boldsymbol{\Theta}$ of the coefficients of H_{Ising} so as to penalize the solutions already visited and prevent a redundant search in the solution space. The matrix is constructed in additive way using the set of k already visited (worst) solutions $\{\boldsymbol{z}_j\}_{j=1,\ldots,k}$ with the help of a function

$$\boldsymbol{m}(\boldsymbol{z}) = \boldsymbol{z}\boldsymbol{z}^T - \boldsymbol{I}_n + \text{diag}(\boldsymbol{z}), \quad \boldsymbol{z} \in \mathcal{Z}, \tag{5.5}$$

where $\text{diag}(\boldsymbol{z})$ constructs a diagonal matrix from a vector $\boldsymbol{z}$. Note that, by construction, the matrix $\boldsymbol{m}(\boldsymbol{z})$ is symmetric and, moreover,

$$\mathsf{E}(\boldsymbol{m}(\boldsymbol{z}), \boldsymbol{z}) > 0, \quad \boldsymbol{z} \in \mathcal{Z}. \tag{5.6}$$

The matrix $\boldsymbol{S}$ is then constructed as the sum:

$$\boldsymbol{S} = \sum_{j=1}^{k} \boldsymbol{m}(\boldsymbol{z}_j). \tag{5.7}$$

Due to (5.6), the tabu matrix $\boldsymbol{S}$ introduces *energetic penalties* on the solutions

$\{\boldsymbol{z}_j\}_{j=1,\ldots,k}$ in the spectrum of the Hamiltonian $H_{\text{Ising}}(\boldsymbol{\Theta} + \boldsymbol{S})$.

Following the key idea of QALS, the TEQAHQO scheme is based on an iterative procedure of candidate solutions generation by reaching and measuring the ground state by QA of a parameter dependent problem Hamiltonian of the form

$$H_P^{(\boldsymbol{\alpha})}(\boldsymbol{S}) = H^{(\boldsymbol{\alpha})} + H_{\text{Ising}}(\boldsymbol{S}), \tag{5.8}$$

where $H^{(\boldsymbol{\alpha})}$ is a generic (arbitrary) Hamiltonian depending on the parameters $\boldsymbol{\alpha}$ which are updated along with the candidate solutions and H_{Ising} is the Ising Hamiltonian used to enable the solution penalties by means of the tabu matrix $\boldsymbol{S}$. To solve the problem by QA, we should have a spin glass described by an Ising Hamiltonian on which we can act with controlled external fields in order to implement $H^{(\boldsymbol{\alpha})}$, i.e. if

$$H^{(\boldsymbol{\alpha})} = H_{\text{Ising}}(\boldsymbol{\alpha}),$$

then we recover the QALS scheme and

$$H_P^{(\boldsymbol{\alpha})}(\boldsymbol{S}) = H_{\text{Ising}}(\boldsymbol{\alpha} + \boldsymbol{S}).$$

In such a case, $\boldsymbol{\alpha} \equiv \boldsymbol{\Theta}$ where $\boldsymbol{\Theta}$ is as given in (5.3).

Within TEQAHQO scheme, the search is simultaneously performed among the candidate solutions $\boldsymbol{z}$ (to find the optimum) and in the space of the parameters $\boldsymbol{\alpha}$ of the Hamiltonian $H^{(\boldsymbol{\alpha})}$ (to find the best representation of the problem in the space of parameterized Hamiltonians). The candidate solutions are obtained by means of the QA, and suboptimal acceptance of the candidate solutions is allowed at the classical counterpart of the algorithm, so the classical part of the hybrid algorithm presents a SA-like structure. The rejected solutions are used to update the tabu matrix $\boldsymbol{S}$ and new candidate solutions are generated, while the parameters $\boldsymbol{\alpha}$ of the Hamiltonian $H^{(\boldsymbol{\alpha})}$ are perturbed in temperature-dependent way, with decreasing temperature parameter. Here, we consider a randomized temperature-dependent function ξ_T to modify the Hamiltonian parameters within the iterative structure of the proposed algorithm TEQAHQO.

The TEQAHQO scheme is presented in Algorithm 5. The QA procedure is run with parameters that are initialized by the randomized function ζ depending on the objective function f of the problem (lines 2, 3, 4), and the worst solution $\boldsymbol{z}'$ is used to initialize the tabu matrix according to (5.7) (line 9). In the iterative structure, the temperature T is constant for a cycle of N iterations (line 13) and decreases according to the function

h and the rate η at any N-th cycle (line 14). For instance, if $h(\eta, T) = T - \eta$ then T decreases linearly, if $h(\eta, T) = T - \eta T$ then T decreases exponentially. At each iteration (line 16) one of the two alternatives is taken, either (with probability q) new parameters $\boldsymbol{\alpha}$ are generated perturbing $\boldsymbol{\alpha}^\star$ corresponding to the current best candidate solution using a temperature-dependent modification function ξ_T which explicitly depends on f, or (with probability $1-q$) $\boldsymbol{z}'$ is sampled using a quantum random number generator, e.g. by measurement on the quantum superposed state. In the former case, the Hamiltonian $H_P^{(\boldsymbol{\alpha})}(\boldsymbol{S})$ is initialized and the QC procedure produces the new candidate solution $\boldsymbol{z}'$ (line 16) that is tested (line 18-19). If $\boldsymbol{z}'$ is better then it is accepted and the parameters are updated accordingly (line 20), else $\boldsymbol{z}'$ is rejected or accepted as a suboptimal solution using SA-like scheme (line 23). Consequently, the tabu matrix is updated in either case (line 25). The Algorithm 5 terminates either if the maximum number i_{max} of iterations is achieved, or the convergence to a solution of the optimization problem is stated (line 30).

Since the TEQAHQO is iteratively applied, the convergence of the procedure to the optimal solution $\boldsymbol{z}^\star$ is usually proved by considering the convergence of the sequence of steady-state probabilities of non-homogeneous Markov chains modeling the sequences of candidate solutions obtained. However, due to (5.7), proving the Markovian nature of such a process is problematic, since the tabu matrix holds virtually unlimited memory of previous states. Thus, it is important to study the collisions in the tabu matrix $\boldsymbol{S}$. We address this issue in the next section.

5.2 Theoretical Convergence

The convergence proof of the TEQAHQO scheme is based on some properties of the SA-like structure implemented by the classical part of Algorithm 5. In order to prove the algorithm convergence we consider the stochastic sequence of the candidate solutions obtained in the algorithm run. At the same time, we need to show finiteness of memory of the tabu matrix $\boldsymbol{S}$.

In order to define the tabu matrices corresponding to the run of TEQAHQO algorithm, let us enumerate the state space. Let the n-dimensional column vector $\boldsymbol{z}^{(i)} \in \mathcal{Z}$ be i-th element of the lexicographically ordered state space $\mathcal{Z}$ (this may be constructively defined as the n-digit binary representation of i in the binary alphabet $\{-1, 1\}$).

Consider now two independent runs of TEQAHQO algorithm. For each such a run encode the sequence of rejected solutions by a nonnegative integer sequence of length 2^n (irregardless of the order in which they appeared in the trajectory). For each such

a sequence $\boldsymbol{a} \in \mathbb{Z}_+^{2^n}$ construct the matrix function $\boldsymbol{S}(\boldsymbol{a})$ using (5.7) and (5.5) in the following way:

$$\boldsymbol{S}(\boldsymbol{a}) = \sum_{i=1}^{2^n} a_i \, \boldsymbol{m}\left(\boldsymbol{z}^{(i)}\right). \tag{5.9}$$

The *collision* in the tabu matrix happens if $\boldsymbol{S}(\boldsymbol{a}) = \boldsymbol{S}(\boldsymbol{b})$ for $\boldsymbol{a} \neq \boldsymbol{b} \in \mathbb{Z}_+^{2^n}$. Such collisions are solutions of a linear system

$$\sum_{i=1}^{2^n} x_i \, \boldsymbol{m}\left(\boldsymbol{z}^{(i)}\right) = \boldsymbol{O}, \tag{5.10}$$

where $\boldsymbol{O}$ is the square zero matrix of dimension n and $x_i := a_i - b_i$ are the variables. Rewriting (5.10) in a vector-matrix form, obtain

$$\boldsymbol{M}\boldsymbol{x} = \boldsymbol{0}, \tag{5.11}$$

where $\boldsymbol{x} \in \mathbb{Z}^{2^n}$ is the (column) vector of componentwise differences of two trajectories and $\boldsymbol{M}$ is the $\frac{n(n+1)}{2} \times 2^n$ matrix containing columnwise the upper-triangular elements of matrices $\boldsymbol{m}(\boldsymbol{z}^{(i)})$, starting from the diagonal elements. As such, solution of the system (5.11) is a vector in the kernel of (5.9). In particular, if $\boldsymbol{x} \in \mathbb{Z}_+^{2^n}$, then a sequence of states corresponding to $\boldsymbol{x}$ produces a zero tabu matrix after a non-zero number of iterations of the QA algorithm, i.e. $\boldsymbol{S}(\boldsymbol{x}) = \boldsymbol{O}$. Let us formally study the solutions of the system (5.11).

Now we recursively define a sequence of matrices, which allows us to establish the desired algebraic properties. Define the following $2^k \times 2^{k-1}$ matrices for any $k \geq 1$:

$$\mathbb{S}^{(k)} := \begin{bmatrix} \mathbb{I}_{2^{k-1}} \\ \mathbb{J}_{2^{k-1}} \end{bmatrix}, \qquad \mathbb{A}^{(k)} := \begin{bmatrix} -\mathbb{I}_{2^{k-1}} \\ \mathbb{J}_{2^{k-1}} \end{bmatrix}, \tag{5.12}$$

where $\mathbb{I}_{2^{k-1}}$ and $\mathbb{J}_{2^{k-1}}$ are the identity matrix and the exchange matrix (matrix with unit vector over antidiagonal) of order 2^{k-1} respectively (conventionally $\mathbb{I}_0 = \mathbb{J}_0 = 1$). Construct the class $SA(n)$ of $2^n \times 1$ matrices having the following form

$$F^{(n)} F^{(n-1)} \cdots F^{(1)} \in SA(n),$$

where $F^{(k)}$ can be either $\mathbb{S}^{(k)}$ or $\mathbb{A}^{(k)}$. In the following proposition we establish group properties of $SA(n)$.

Proposition 1. *The following properties of $SA(n)$ are true for any $n \geqslant 1$:*

1. *With the exception of* $\mathbb{S}^{(n)}\cdots\mathbb{S}^{(1)}$*, any element of* $SA(n)$ *contains the same number of* -1 *and* 1*, equal to* 2^{n-1}*.*

2. $SA(n)$ *equipped with the Hadamard (componentwise) product* $\circ$ *is an abelian group with the neutral element* $\mathbb{S}^{(n)}\cdots\mathbb{S}^{(1)}$ *and any element being the inverse of itself.*

3. *The elements of* $SA(n)$ *are orthogonal w.r.t. the Euclidean scalar product.*

Proof. In the proof, we use induction on n.

(1) For $n = 1$ the group $SA(1)$ contains only $\mathbb{S}^{(1)}$ and $\mathbb{A}^{(1)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$. Let (1) hold good for $SA(j)$, $j \leq n$. Denote

$$v = F^{(n)}F^{(n-1)}\cdots F^{(1)},$$

where $F^{(k)} \in \{\mathbb{S}^{(k)}, \mathbb{A}^{(k)}\}$ for $k \leq n+1$. Then since

$$\mathbb{S}^{(n+1)}v = \begin{bmatrix} v \\ \mathbb{J}_{2^n}v \end{bmatrix} \text{ and } \mathbb{A}^{(n+1)}v = \begin{bmatrix} -v \\ \mathbb{J}_{2^n}v \end{bmatrix},$$

the statement (1) follows for $n+1$ and the induction step follows.

(2) For $n = 1$, we have

$$\mathbb{S}^{(1)} \circ \mathbb{S}^{(1)}, \mathbb{S}^{(1)} \circ \mathbb{A}^{(1)}, \mathbb{A}^{(1)} \circ \mathbb{S}^{(1)}, \mathbb{A}^{(1)} \circ \mathbb{A}^{(1)} \in SA(1),$$

by definition (5.12). Let (2) hold good for $SA(j)$, $j \leq n$. Denote

$$v = F^{(n)}F^{(n-1)}\cdots F^{(1)},\ w = E^{(n)}E^{(n-1)}\cdots E^{(1)},$$

where $E^{(k)}, F^{(k)} \in \{\mathbb{S}^{(k)}, \mathbb{A}^{(k)}\}$, $k \leq n+1$. Similarly to the proof of statement (1),

$$F^{(n+1)}v = \begin{bmatrix} \pm v \\ \mathbb{J}_{2^n}v \end{bmatrix}, \qquad E^{(n+1)}w = \begin{bmatrix} \pm w \\ \mathbb{J}_{2^n}w \end{bmatrix},$$

and hence

$$F^{(n+1)}v \circ E^{(n+1)}w = \begin{bmatrix} \pm v \circ w \\ \mathbb{J}_{2^n}(v \circ w) \end{bmatrix} \in SA(n+1)$$

holds by definition of $SA(n+1)$, since

$$\begin{bmatrix} v \circ w \\ \mathbb{J}_{2^n}(v \circ w) \end{bmatrix} = \mathbb{S}^{(n+1)}(v \circ w),$$

$$\begin{bmatrix} -v \circ w \\ \mathbb{J}_{2^n}(v \circ w) \end{bmatrix} = \mathbb{A}^{(n+1)}(v \circ w),$$

and finally $v \circ w \in SA(n)$, which completes the induction step. It remains to note that the neutral element is given by $\mathbb{S}^{(n)} \cdots \mathbb{S}^{(1)}$, whose entries are identically 1, as a direct consequence of the definition of Hadamard product, and hence each element of the group is inverse of itself.

(3) As a consequence of (1) and (2) we have that the Euclidean scalar product between two distinct elements v and w of $SA(n)$ is nothing but the sum of the entries of $v \circ w$ that presents an equal number of -1s and 1s. □

Now we consider the matrix $\boldsymbol{M}$ of the linear system (5.11) and prove the following statement.

Proposition 2. *The rows of $\boldsymbol{M}$ are transposed elements of a subset $SA_M \subset SA(n)$.*

Proof. Construct a $2^n \times n$ matrix $\boldsymbol{B}$ by columns B_i, $i = 1, \ldots, n$, using specific vectors from the $SA(n)$ group as follows:

$$B_i = F^{(n)} \ldots F^{(1)}, \quad F^{(i)} = \mathbb{A}^{(i)}, F^{(j)} = \mathbb{S}^{(j)}, j \neq i.$$

This means that $\boldsymbol{B}$ by column consists of the column vectors $\mathbb{A}^{(n)}\mathbb{S}^{(n-1)} \ldots \mathbb{S}^{(1)}$, $\mathbb{S}^{(n)}\mathbb{A}^{(n-1)} \ldots \mathbb{S}^{(1)}$ and so on. However, this process can be done recursively starting from the last, n-th column, as follows. We start with the column vector $\mathbb{A}^{(1)} = \begin{bmatrix} -1 \\ 1 \end{bmatrix}$ of two components. Then we left-multiply it with $\mathbb{S}^{(2)}$ which in fact produces a four-component vector containing the values of $\mathbb{A}^{(1)}$ mirrored, i.e. $(-1, 1, 1, -1)^T$. Now we add from the left another column (which finally becomes the column $n-1$ in the constructed matrix), which contains values $\mathbb{A}^{(2)}\mathbb{S}^{(1)} = (-1, -1, 1, 1)^T$. This means that the original column $\mathbb{A}^{(1)}$ corresponds to the values -1 of the column added from the left, whereas the mirrored column $\mathbb{A}^{(1)}$ corresponds to the values 1 in the left column. These two actions (mirroring, adding from the left a column vector of sequence of -1's followed by 1's of equal length) is repeated until the constructed matrix $\boldsymbol{B}$ has n columns.

It is easy to see that the resulting matrix $\boldsymbol{B}$ contains classical Gray (binary reflected) codes in the binary alphabet $\{-1, 1\}$, according to recursive Gray code generation procedure, see e.g. [69]. It is well known that the Gray codes of length n enumerate all the possible values of 2^n strings in the binary alphabet. As such, there exists a permutation

$$p : \{1, \ldots, 2^n\} \mapsto \{1, \ldots, 2^n\},$$

such that $p(i)$ is the row number in the matrix $\boldsymbol{B}$ of the (classical) binary representation of the value i in the alphabet $\{-1, 1\}$ having length n. From this matrix, we construct an $\frac{n(n+1)}{2} \times 2^n$ matrix $\boldsymbol{A}$ from the transposed matrix $\boldsymbol{B}^T$ followed by pairwise Hadamard products of its rows,

$$B_i \circ B_j, i < j, i, j \in \{1, \ldots, n\}.$$

Note that, by statement (2) of Proposition 1, all rows of $\boldsymbol{A}$ are transposed elements of $SA(n)$. We denote these elements as a subset

$$SA_M \subset SA(n).$$

Recall now the construction of matrix $\boldsymbol{M}$. It contains by columns all the values of diagonal and upper triangular elements of matrices $\boldsymbol{m}(\boldsymbol{z}^{(i)})$ using (5.5), where $\boldsymbol{z}^{(i)}$ is the length-n binary representation of $i = 1, \ldots, 2^n$ in the alphabet $\{-1, 1\}$. As such, i-th column of the matrix $\boldsymbol{M}$ corresponds to $p(i)$-th column of matrix $\boldsymbol{A}$, which completes the proof. □

An immediate consequence of Propositions 1 and 2 is the following Proposition.

Proposition 3. *The space of solutions $\mathcal{V}_M$ of the linear system* (5.11) *is spanned by the elements of $SA(n) \setminus SA_M$ and its dimension is:*

$$\dim \mathcal{V}_M = 2^n - \frac{n(n+1)}{2}.$$

Intuitively this means that any solution $\boldsymbol{x}$ of the system (5.11) is a linear combination of the vectors in $SA(n) \setminus SA_M$ which form a basis being orthogonal w.r.t. Euclidean scalar product, according to (3) of Proposition 1. At the same time, if the solution vector is non-negative, $\boldsymbol{x} \geq \boldsymbol{0}$, then the corresponding tabu matrix $\boldsymbol{S}(\boldsymbol{x})$, constructed according to (5.9), will be a zero matrix. Below we use this result to prove the convergence of TEQAHQO algorithm.

In Section 5.1, we described the candidate solution generation by Algorithm 5.

In what follows, we study the convergence of the TEQAHQO trajectory under two simplifying assumptions:

1. the sequence of candidate solutions forms a non-homogeneous temperature-dependent stochastic sequence in a finite state space $\mathcal{Z}$;

2. the sequence of tabu matrices evolves on a finite (multidimensional) state space $\mathcal{S}$.

The latter assumption may be relaxed, however, it causes additional efforts to prove positive recurrence of the corresponding two-dimensional process describing the TEQAHQO algorithm evaluation.

We consider the two-dimensional discrete-time stochastic process

$$\{\boldsymbol{Z}_i^{(T)}, \boldsymbol{S}_i^{(T)}\}_{i\geq 1} \tag{5.13}$$

living in the state space $\mathcal{Z}\times\mathcal{S}$, where $\boldsymbol{Z}_i^{(T)}$ is the current candidate solution and $\boldsymbol{S}_i^{(T)}$ the tabu matrix at i-th step of the algorithm. We note that the dependence on the current temperature, T, is stressed in the notation, however, the sequence of temperatures is a non-random sequence $\{T_j\}_{j\geq 1}$ which is selected in a non-random way. Now we show that the process (5.13) is regenerative.

Proposition 4. *Algorithm 5 converges.*

Proof. Since the component $\boldsymbol{Z}_i^{(T)}$ of the process (5.13) lives in a finite state space $\mathcal{Z}$, we can build a sequence of random times $t_1, t_2, \ldots$ such that $\boldsymbol{Z}_i^{(T)}$ visits a specific state, say, $\hat{\boldsymbol{z}} \in \mathcal{Z}$. Due to the quantum nature of the computation, the probability of generating the candidate $\hat{\boldsymbol{z}}$ given any current solution $\boldsymbol{z}' \neq \hat{\boldsymbol{z}}$ is uniformly bounded from below by the value $(1-q)/2^n$, where $1-q$ is the probability to obtain the state by quantum random number generator (line 16 of Algorithm 5), in this case the outcome is produced by a uniform sampling on $\mathcal{Z}$. As such, since $\mathcal{Z}$ is finite, geometrical trial argument (see e.g. [11]) allows to conclude that the sequence $\{t_j\}_{j\geq 1}$ is infinite and has finite mean inter-event times.

Using Proposition 3 we take a subsequence $\{t_{j_k}\}_{k\geq 1}$ such that $\boldsymbol{S}_{t_{j_k}}^{(T)} = \boldsymbol{O}$. The probability of obtaining such a matrix from any other matrix, say, S', is bounded from below by positive value. Indeed, due to the fact that there are many solutions of the system (5.11), we select the closest non-negative solution $\boldsymbol{x} \geq \boldsymbol{0}$ such that $\boldsymbol{x} \geq \boldsymbol{y}$, where $\boldsymbol{y}$ is any trajectory that gives a tabu matrix S'. Due to assumption of finiteness of the state space $\mathcal{S}$, the maximal distance between the vectors $\boldsymbol{x}$ and $\boldsymbol{y}$, that is,

$\max_{k=1,\ldots,2^n}|x_k - y_k|$, is finite. Thus, the number of steps to reach $\boldsymbol{x}$ from $\boldsymbol{y}$ by adding the solutions to tabu matrix, is uniformly bounded from above. Finally, we conclude that the matrix $\boldsymbol{S}^{(T)}_{t_{j_k}}$ regenerates at zero with positive probability, and using the geometrical trial argument, conclude that the subsequence $\{t_{j_k}\}_{k\geq 1}$ has finite mean. Thus, overall the process (5.13) is a positive recurrent regenerative process which guarantees the convergence of Algorithm 5. □

5.3 Numerical Illustration of Convergence

In this section, we numerically illustrate the theoretical results related to the convergence of TEQAHQO scheme. Namely, we use a simplified version of the QALS algorithm, which is a particular case of the TEQAHQO scheme, to study the collisions in the tabu matrix. We do so by solving a simple QUBO problem, firstly by performing simulation and secondly by running the minimization in hybrid regime on QA hardware. In both experiments, we watch the tabu matrix and observe the iteration at which the matrix first becomes zero.

The specific QUBO problem was the minimization of the quadratic function $f(\boldsymbol{x}) = \boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x}$ having the following matrix

$$\boldsymbol{Q} = \begin{bmatrix} 0 & -0.2 & 0 & 0 \\ 0 & 1.0 & -0.5 & 1.0 \\ 0 & 0 & -0.9 & 0 \\ 0 & 0 & 0 & 0.7 \end{bmatrix}.$$

This results in the following function to be minimized:

$$f(\boldsymbol{x}) = -0.2x_1x_2 + x_2^2 - 0.5x_2x_3 + x_2x_4 - 0.9x_3^2 + 0.7x_4^2,$$

where the variables $x_i \in \{-1, 1\}, i = 1, \ldots, 4$. This function has minima at $\boldsymbol{x} = (-1, -1, -1, 1)$ and, symmetrically, at $-\boldsymbol{x}$, with the value of -0.9.

We use the following configuration of the parameters related to Algorithm 5:

$$i_{max} = 200,\ N_{max} = 100,\ q = 0.99,\ \eta = 0.2.$$

In the considered version of QALS (introduced in [90]), the role of temperature parameter is played by a sequence of probabilities $\{p_i\}_{i\geq 1}$ that decrease geometrically with i from $p_1 = 1$ to the fixed value $p_\delta = 0.01$, and the temperature parameter T is related

to (a generic member of the sequence) p by the following relation

$$p - p_\delta = e^{-\frac{1}{T}}. \tag{5.14}$$

Since the recursive relation to obtain the sequence of probabilities $\{p_i\}_{i\geq 1}$ in [90] was

$$p_{i+1} = p_i - (p_i - p_\delta)\eta, \quad , i \geq 1,$$

the temperature modification function for the algorithm can be easily deduced from (5.14) as

$$h(\eta, T) = \frac{T}{1 - T\log(1-\eta)}.$$

Moreover, due to (5.14), the initial value T_{max} is given as

$$T_{max} = -\left(\log(0.99)\right)^{-1}.$$

Finally, the initialization ζ was a uniform (in $[0, 1]$) random number generator and the modification ξ_T was the identity function.

In the simulation experiment we used 10000 repeated runs of the algorithm. Among these, in 457 trajectories there was at least one collision registered in which the matrix $\boldsymbol{S}$ was identically zero, and the histogram is built for the trajectory length (in terms of the number of iterations i) until the first zero (after a zero appeared, iterations were cancelled). The corresponding empirical distribution is depicted in Figure 5.1 (top).

In the experiment on a hardware, we used 250 repeated runs of length 200 each, at Quantum hardware which was D-Wave computer (Advantage system 4.1, 5627 qubits and 40279 couplers, Pegasus topology). Among these, 15 zeroes were obtained, which gives approximately the same rate as in simulation (6% compared to 4.57%, respectively). These results required approximately 4.383 minutes of computing time at QPU. The corresponding empirical distribution is shown in Figure 5.1 (bottom).

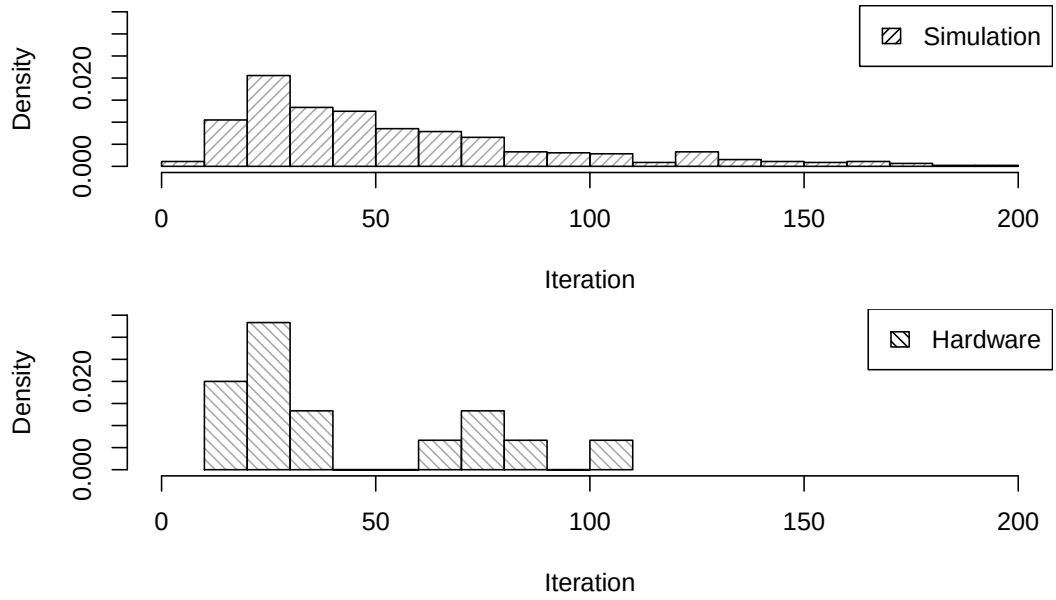


Figure 5.1: The histogram of the number of steps before the first appearance of a zero tabu matrix starting from a zero matrix at the time origin.

5.4 Quantum Circuit Optimization using TEQAHQO Scheme

To optimize a QC using TEQAHQO, we propose to modify the QC optimization approach introduced in Chapter 4 by using the TEQAHQO scheme instead of original QA at Step 3.3 in Section 4.2. For this, we construct the QUBO matrix Q corresponding to the QC optimization problem instance and replace objective function f to be minimized with $\mathbf{x}^T Q \mathbf{x}$ in Algorithm 5 where $\mathbf{x} = \{0, 1\}^n$.

As an example QC to optimize, we consider the Toffoli gate illustrated in Figure 5.2. In order to be LNN-compliant, 2 $SWAP$s are needed if initial linear qubit placement follows $\begin{bmatrix} q_0 & q_1 & q_2 \end{bmatrix}$.

Note that in Figure 5.3, the dependency of the gates of circuit from Figure 5.2 is illustrated.

To optimize the circuit using TEQAHQO scheme, we use the following configuration of the parameters related to Algorithm 5:

$d_{min} = 70,\ \eta = 0.02,\ i_{max} = 100,\ k = 1000,\ \lambda_0 = 3/2,\ N = 10,\ N_{max} = 80,\ p_\delta = 0.01,\ q = 0.9.$

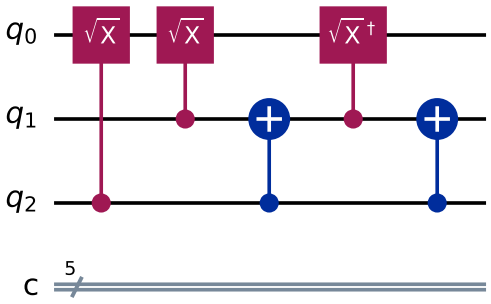


Figure 5.2: Toffoli gate.

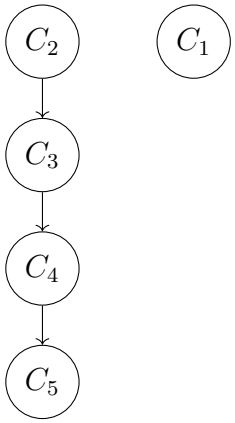


Figure 5.3: Dependency of gates in implementation of Toffoli gate from Figure 5.2.

5.4.1 Numerical Results while Optimizing Toffoli Gate in LNN Architecture

At first, we optimize the circuit using Simulated Annealing. After applying the proposed approach, one $SWAP$ is needed to make it LNN-compliant, at best. During optimization, the circuit is divided into 2 NN-compliant subcircuits: the first one contains the first two gates and the other gates form the second subcircuit (see Figure 5.4). The qubits are placed following $A^1 = \begin{bmatrix} q_1 & q_0 & q_2 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_0 & q_1 & q_2 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively.

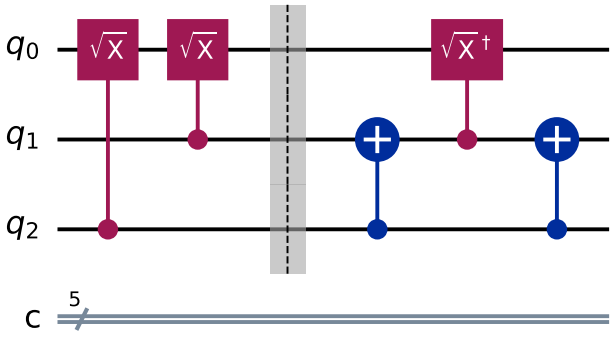


Figure 5.4: Toffoli gate optimized.

While using QA, it divides Toffoli gate into the same subcircuits as SA does (see Fig-

ure 5.4) with the following qubit placements $A^1 = \begin{bmatrix} q_2 & q_0 & q_1 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_2 & q_1 & q_0 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. One $SWAP$ is used to make them consistent (by swapping q_0 and q_1.)

5.4.2 Numerical Results while Optimizing Toffoli Gate in 2D NN Architecture

For qubit placement defined by $A = \begin{bmatrix} q_0 & q_1 \\ q_2 & - \end{bmatrix}$ in 2D NN architecture, Toffoli gate needs 4 $SWAP$s. While optimizing it, SA divides it in the same way (see Figure 5.4) with qubit placements $A^1 = \begin{bmatrix} q_2 & q_0 \\ - & q_1 \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_1 & q_0 \\ q_2 & - \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. 2 $SWAP$s are used to make them consistent (by placing q_1 to bottom-left and then swapping it with q_2,) whereas initially 4 $SWAP$s are required.

While optimizing the QC using QA, final subcircuits are the same ones (see Figure 5.4) with qubit placements $A^1 = \begin{bmatrix} q_0 & q_1 \\ q_2 & - \end{bmatrix}$ and $A^2 = \begin{bmatrix} q_0 & q_1 \\ - & q_2 \end{bmatrix}$ within the 1st and the 2nd subcircuits, respectively. Only 1 $SWAP$ is needed to make them consistent (by placing q_2 to bottom-right).

Note that we used 1000 repeated runs of length 100 each, on quantum hardware which was a D-Wave computer of version `Advantage system 4.1`. The software is available through link `https://github.com/MakarMariia97/Quantum-Annealing-for-solving-QUBO-Problems`.

5.5 Discussion

We introduce the metaheuristic TEQAHQO scheme and outline the approach for formally proving its convergence. This scheme incorporates a tabu matrix structure within an Ising model framework. We clarify the role of the matrix in establishing the asymptotic convergence by discussing the regeneration of the Markov process that describes the computation. Furthermore, we empirically validate the existence of this regeneration phenomenon in a specific case. However, the speed of convergence and strategies for improving the algorithm's efficiency warrant separate investigation. One potential direction for further research is the parameterization of the tabu matrix aiming to find a balance between the depth of dependency and the speed of convergence in the optimization algorithm. Additionally, a comparative analysis with other existing hybrid

quantum computing schemes could provide valuable insights. Finally, we apply the TEQAHQO scheme to the QC optimization problem and our numerical experiments demonstrate its potential effectiveness.

Chapter 6

Discussion and Conclusion

In this thesis, we investigate the QC optimization problem with the goal of minimizing the SWAP count as the optimization criterion. We begin by conducting a comprehensive analysis of the state of the art related to this problem. From the overview, we identify two promising QC optimization approaches based on qubit reordering: one emphasizing global reordering and the other focusing on local reordering.

Building on these foundational approaches, we propose two innovative quantum-based QC optimization strategies that utilize QA as an optimization solver for the first time in this context. The first approach aims to optimize quantum circuits in a linear NN architecture by implementing global qubit reordering. This method is grounded in the formulation of a GP problem, which effectively captures the underlying structure of the optimization task. By considering the QC as a graph, we can systematically reorganize the qubits to minimize the SWAP count by solving GP problem using QA, paving the way for more efficient quantum computations.

The second approach employs Boolean satisfiability theory to facilitate local qubit reordering. This strategy is used for optimizing QCs within a 2D NN architecture, which is a common configuration in quantum hardware. By defining local reordering in a way that directly addresses the constraints imposed by the architecture, we can achieve significant reductions in SWAP counts without compromising the fidelity of the quantum operations.

To demonstrate the effectiveness of our proposed methods, we conduct extensive numerical experiments on a range of quantum circuits of varying sizes. The results reveal that our optimization strategies yield significant improvements, showcasing performance that is competitive with those achieved by classical solvers. These findings underscore the potential of quantum-based approaches in addressing QC optimization.

Furthermore, we explore the application of QA enhanced with a tabu mechanism to improve the optimization process based on local qubit reordering. This mechanism introduces a form of memory into the optimization process, preventing the revisitation of previously explored configurations and thereby accelerating convergence toward optimal solutions. To establish the robustness and applicability of this approach, we also provide a proof of convergence based on Markov process regeneration, ensuring that our method consistently leads to optimal or near-optimal solutions over time. The experiments performed illustrate perspectivity of using tabu-enhanced QA for QC optimization both for linear and 2D NN architecture.

In conclusion, this thesis not only contributes new methodologies for quantum circuit optimization but also lays the groundwork for future research (meaning increase of the dimension of topology considered). By leveraging the unique capabilities of quantum algorithms and rigorously validating their performance, we aim to advance the field of quantum computing, ultimately paving the way for more efficient implementations on real-world quantum hardware.

Bibliography

[1] Estimating quantum volume for Advantage. https://support.dwavesys.com/hc/en-us/community/posts/360051945133-Estimating-Quantum-Volume-for-Advantage. [Online; accessed 18-January-2025].

[2] Graph Partitioning. https://github.com/dwave-examples/graph-partitioning. [Online; accessed 18-January-2025].

[3] Minor-Embedding. https://docs.ocean.dwavesys.com/en/stable/concepts/embedding.html#embedding-sdk. [Online; accessed 18-January-2025].

[4] Welcome to qubovert's documentation! https://qubovert.readthedocs.io/en/latest/. Accessed: 2025-04-21.

[5] Rajeev Acharya, Igor Aleiner, Richard Allen, Trond I. Andersen, Markus Ansmann, Frank Arute, Kunal Arya, Abraham Asfaw, Juan Atalaya, Ryan Babbush, Dave Bacon, Joseph C. Bardin, Joao Basso, Andreas Bengtsson, Sergio Boixo, Gina Bortoli, Alexandre Bourassa, Jenna Bovaird, Leon Brill, Michael Broughton, Bob B. Buckley, David A. Buell, Tim Burger, Brian Burkett, Nicholas Bushnell, Yu Chen, Zijun Chen, Ben Chiaro, Josh Cogan, Roberto Collins, Paul Conner, William Courtney, Alexander L. Crook, Ben Curtin, Dripto M. Debroy, Alexander Del Toro Barba, Sean Demura, Andrew Dunsworth, Daniel Eppens, Catherine Erickson, Lara Faoro, Edward Farhi, Reza Fatemi, Leslie Flores Burgos, Ebrahim Forati, Austin G. Fowler, Brooks Foxen, William Giang, Craig Gidney, Dar Gilboa, Marissa Giustina, Alejandro Grajales Dau, Jonathan A. Gross, Steve Habegger, Michael C. Hamilton, Matthew P. Harrigan, Sean D. Harrington, Oscar Higgott, Jeremy Hilton, Markus Hoffmann, Sabrina Hong, Trent Huang, Ashley Huff, William J. Huggins, Lev B. Ioffe, Sergei V. Isakov, Justin Iveland, Evan Jeffrey, Zhang Jiang, Cody Jones, Pavol Juhas, Dvir Kafri, Kostyantyn

Kechedzhi, Julian Kelly, Tanuj Khattar, Mostafa Khezri, Mária Kieferová, Seon Kim, Alexei Kitaev, Paul V. Klimov, Andrey R. Klots, Alexander N. Korotkov, Fedor Kostritsa, John Mark Kreikebaum, David Landhuis, Pavel Laptev, Kim-Ming Lau, Lily Laws, Joonho Lee, Kenny Lee, Brian J. Lester, Alexander Lill, Wayne Liu, Aditya Locharla, Erik Lucero, Fionn D. Malone, Jeffrey Marshall, Orion Martin, Jarrod R. McClean, Trevor McCourt, Matt McEwen, Anthony Megrant, Bernardo Meurer Costa, Xiao Mi, Kevin C. Miao, Masoud Mohseni, Shirin Montazeri, Alexis Morvan, Emily Mount, Wojciech Mruczkiewicz, Ofer Naaman, Matthew Neeley, Charles Neill, Ani Nersisyan, Hartmut Neven, Michael Newman, Jiun How Ng, Anthony Nguyen, Murray Nguyen, Murphy Yuezhen Niu, Thomas E. O'Brien, Alex Opremcak, John Platt, Andre Petukhov, Rebecca Potter, Leonid P. Pryadko, Chris Quintana, Pedram Roushan, Nicholas C. Rubin, Negar Saei, Daniel Sank, Kannan Sankaragomathi, Kevin J. Satzinger, Henry F. Schurkus, Christopher Schuster, Michael J. Shearn, Aaron Shorter, Vladimir Shvarts, Jindra Skruzny, Vadim Smelyanskiy, W. Clarke Smith, George Sterling, Doug Strain, Marco Szalay, Alfredo Torres, Guifre Vidal, Benjamin Villalonga, Catherine Vollgraff Heidweiller, Theodore White, Cheng Xing, Z. Jamie Yao, Ping Yeh, Juhwan Yoo, Grayson Young, Adam Zalcman, Yaxing Zhang, and Ningfeng Zhu. Suppressing quantum errors by scaling a surface code logical qubit. *Nature*, 614(7949):676–681, February 2023.

[6] D. Aharonov, W. van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic Quantum Computation is equivalent to standard quantum computation. In *45th Annual IEEE Symposium on Foundations of Computer Science*, pages 42–51, 2004.

[7] Mohammad Gh. Alfailakawi, Imtiaz Ahmad, and Suha Hamdan. Harmony-Search algorithm for 2D Nearest Neighbor quantum circuits realization. *Expert Systems with Applications*, 61(C):16–27, November 2016.

[8] M. Amy, D. Maslov, M. Mosca, and M. Roetteler. A Meet-in-the-Middle algorithm for fast synthesis of depth-optimal quantum circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(6):818–830, 2013. Publisher: Institute of Electrical and Electronics Engineers (IEEE).

[9] Simon Apers, Andrés Gilyén, and Stacey Jeffery. A unified framework of quantum walk search. *Leibniz International Proceedings in Informatics*, page 13, 2021.

[10] Mona Arabzadeh, Morteza Saheb Zamani, Mehdi Sedighi, and Mehdi Saeedi. Depth-optimized reversible circuit synthesis. *Quantum Information Processing*, 12(4):1677–1699, April 2013.

[11] Søren Asmussen. *Applied probability and queues*. Stochastic Modelling and Applied Probability. Springer, New York, 2003.

[12] Radhakrishnan Balu, Chaobin Liu, and Salvador E. Venegas-Andraca. Probability distributions for Markov chains based quantum walks. *Journal of Physics A: Mathematical and Theoretical*, 51(3):035301, January 2018. arXiv: 1703.04131.

[13] Anindita Banerjee and Anirban Pathak. New designs of reversible sequential devices. 08 2009.

[14] Adriano Barenco, Charles H. Bennett, Richard Cleve, David P. DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A. Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical Review A*, 52(5):3457–3467, November 1995.

[15] M. Barreto, G. Abal, and S. Nesmachnow. A parallel spatial quantum search algorithm applied to the 3-SAT problem. In *In Proceedings of XII Argentine Symposium on Artificial Intelligence*, 2011.

[16] Anirban Bhattacharjee, Chandan Bandyopadhyay, Angshu Mukherjee, Robert Wille, Rolf Drechsler, and Hafizur Rahaman. An ant colony based mapping of quantum circuits to Nearest Neighbor architectures. *Integration*, 78:11–24, May 2021.

[17] Anirban Bhattacharjee, Chandan Bandyopadhyay, Robert Wille, Rolf Drechsler, and Hafizur Rahaman. Improved look-ahead approaches for Nearest Neighbor synthesis of 1D quantum circuits. In *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*, pages 203–208, January 2019.

[18] Zhengbing Bian, Fabian Chudak, William Macready, Aidan Roy, Roberto Sebastiani, and Stefano Varotti. Solving SAT (and MaxSAT) with a quantum annealer: Foundations, encodings, and preliminary results. *Information and Computation*, 275:104609, 2020.

[19] Enrico Blanzieri, Davide Pastorello, Valter Cavecchia, Alexander Rumyantsev, and Mariia Maltseva. Evaluating the convergence of tabu enhanced hybrid quantum optimization. *Quantum Information Processing*, 22(5), May 2023.

[20] Hamidreza Bolhasani and Amir Rahmani. An introduction to quantum computers architecture. January 2019.

[21] Costantino Carugno, Maurizio Ferrari Dacrema, and Paolo Cremonesi. Evaluating the job-shop scheduling problem on a D-Wave quantum annealer. *Scientific Reports*, 12(1), April 2022.

[22] Davide Castelvecchi. IBM releases first-ever 1,000-qubit quantum chip. *Nature*, 624(7991):238–238, December 2023.

[23] Amlan Chakrabarti and Susmita Sur-Kolay. Rules for synthesizing quantum boolean circuits using minimized Nearest-Neighbour templates. In *15th International Conference on Advanced Computing and Communications (ADCOM 2007)*, pages 183–189, Guwahati, Assam, India, December 2007. IEEE.

[24] Amlan Chakrabarti, Susmita Sur-Kolay, and Ayan Chaudhury. Linear Nearest Neighbor synthesis of reversible circuits by Graph Partitioning, 2011.

[25] Lalengmawia Chhangte and Alok Chakrabarty. Near-optimal circuit mapping with reduced search paths on IBM quantum architectures. *Microprocessors and Microsystems*, 94:104637, October 2022.

[26] Andrew M. Childs and Wim van Dam. Quantum algorithms for algebraic problems. *Reviews of Modern Physics*, 82(1):1–52, January 2010.

[27] Alexander Cowtan, Silas Dilkes, Ross Duncan, Will Simmons, and Seyon Sivarajah. Phase gadget synthesis for shallow circuits. *Electronic Proceedings in Theoretical Computer Science*, 318:213–228, May 2020.

[28] Maria Luisa Dalla Chiara, Roberto Giuntini, and Giuseppe Sergioli. A quantum approach to Pattern Recognition and Machine Learning. Part II. *International Journal of Theoretical Physics*, 63(2):1–10, February 2024.

[29] Kamalika Datta, Bhadreswar Ghuku, Devi Sandeep, Indranil Sengupta, and Hafizur Rahaman. A cycle based reversible logic synthesis approach. In *2013 Third International Conference on Advances in Computing and Communications*, pages 316–319, 2013.

[30] Kamalika Datta, Gaurav Rathi, Indranil Sengupta, and Hafizur Rahaman. Synthesis of reversible circuits using heuristic search method. In *2012 25th International Conference on VLSI Design*, pages 328–333, Hyderabad, India, January 2012. IEEE.

[31] Marc G. Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. Towards optimal topology aware quantum circuit synthesis. In *2020 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 223–234, Denver, CO, USA, October 2020. IEEE.

[32] Marc Grau Davis, Joaquin Chung, Dirk Englund, and Rajkumar Kettimuthu. Towards distributed quantum computing by qubit and gate Graph Partitioning techniques, 2023.

[33] Marc Grau Davis, Ethan Smith, Ana Tudor, Koushik Sen, Irfan Siddiqi, and Costin Iancu. Heuristics for quantum compiling with a continuous gate set, December 2019. arXiv:1912.02727 [quant-ph].

[34] Timothée Goubault de Brugière. *Methods for optimizing the synthesis of quantum circuits. (Méthodes pour l'optimisation de la synthèse de circuits quantiques)*. PhD thesis, University of Paris-Saclay, France, 2020.

[35] D. de Falco, B. Apolloni, and Nicolò Cesa-Bianchi. *A numerical implementation of Quantum Annealing*. Rapporto interno. University, Forschungszentrum BiBoS Bielefeld-Bochum Stochastik, July 1988.

[36] Alexis De Vos and Stijn De Baerdemacker. Block-ZXZ synthesis of an arbitrary quantum circuit. *Physical Review A*, 94(5):052317, November 2016.

[37] docs.dwavesys.com. What is Quantum Annealing?, 2021.

[38] Ghizlane Echbarthi and Hamamache Kheddouci. Fractional greedy and partial restreaming partitioning: New methods for massive Graph Partitioning. In *2014 IEEE International Conference on Big Data (Big Data)*, pages 25–32, October 2014.

[39] Stefan Edelkamp and Stefan Schrödl. Chapter 13 - constraint search. In Stefan Edelkamp and Stefan Schrödl, editors, *Heuristic Search*, pages 571–631. Morgan Kaufmann, San Francisco, 2012.

[40] Suguru Endo, Zhenyu Cai, Simon C. Benjamin, and Xiao Yuan. Hybrid quantum-classical algorithms and quantum error mitigation. *Journal of the Physical Society of Japan*, 90(3):032001, March 2021.

[41] Guy Even, Joseph Seffi Naor, Satish Rao, and Baruch Schieber. Divide-and-Conquer approximation algorithms via spreading metrics. *Journal of the ACM (JACM)*, 47(4):585–616, July 2000.

[42] Ulrich Faigle and Walter Kern. Some convergence results for probabilistic tabu search. *ORSA Journal on Computing*, 4(1):32–37, February 1992.

[43] Azim Farghadan and Naser Mohammadzadeh. Quantum circuit physical design flow for 2D Nearest Neighbor architectures: Quantum circuit design flow. *International Journal of Circuit Theory and Applications*, 45, March 2017.

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

[45] Benedikt Fauseweh. Quantum many-body simulations on digital quantum computers: State-of-the-art and future challenges. *Nature Communications*, 15(1), March 2024.

[46] Daniel Foead, Alifio Ghifari, Marchel Budi Kusuma, Novita Hanafiah, and Eric Gunawan. A systematic literature review of A* pathfinding. *Procedia Computer Science*, 179:507–514, 2021.

[47] Bennett L. Fox. Integrating and accelerating tabu search, Simulated Annealing, and genetic algorithms. *Annals of Operations Research*, 41(2):47–67, June 1993.

[48] M.R. Garey, D.S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, February 1976.

[49] Laura Gentini, Alessandro Cuccoli, Stefano Pirandola, Paola Verrucchi, and Leonardo Banchi. Noise-resilient variational hybrid quantum-classical optimization. *Physical Review A*, 102(5):052414, November 2020.

[50] Fred Glover. Tabu search and finite convergence. *Discrete Applied Mathematics*, 119(1):3–36, 2002. DOI: 10.1016/S0166-218X(01)00263-3.

[51] Kapil Goswami, Gagan Anekonda Veereshi, Peter Schmelcher, and Rick Mukherjee. Solving the Travelling Salesman Problem using a single qubit, 2024.

[52] T. M. Graham, Y. Song, J. Scott, C. Poole, L. Phuttitarn, K. Jooya, P. Eichler, X. Jiang, A. Marra, B. Grinkemeyer, M. Kwon, M. Ebert, J. Cherek, M. T. Lichtman, M. Gillette, J. Gilbert, D. Bowman, T. Ballance, C. Campbell, E. D. Dahl, O. Crawford, N. S. Blunt, B. Rogers, T. Noel, and M. Saffman. Multi-qubit entanglement and algorithms on a neutral-atom quantum computer. *Nature*, 604(7906):457–462, April 2022.

[53] D. Grosse, R. Wille, G.W. Dueck, and R. Drechsler. Exact Multiple-Control Toffoli network synthesis with SAT techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(5):703–715, May 2009.

[54] Daniel Grosse, Robert Wille, Gerhard W. Dueck, and Rolf Drechsler. Exact synthesis of elementary quantum gate circuits for reversible functions with don't cares. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 214–219, Dallas, TX, May 2008. IEEE.

[55] Darrall Henderson, Sheldon H. Jacobson, and Alan W. Johnson. The theory and practice of Simulated Annealing. In Fred Glover and Gary A. Kochenberger, editors, *Handbook of Metaheuristics*, volume 57, pages 287–319. Springer US, Boston, MA, 2003. DOI: 10.1007/0-306-48056-5_10.

[56] Mahboobeh Houshmand, Mehdi Sedighi, Morteza Saheb Zamani, and Kourosh Marjoei. Quantum circuit synthesis targeting to improve One-Way quantum computation pattern cost metrics. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13:1–27, 2017.

[57] Ciaran Hughes, Joshua Isaacson, Anastasia Perry, Ranbel F. Sun, and Jessica Turner. *Quantum Computing for the Quantum Curious*. Springer International Publishing, Cham, 2021.

[58] Raban Iten, Roger Colbeck, Ivan Kukuljan, Jonathan Home, and Matthias Christandl. Quantum circuits for isometries. *Physical Review A*, 93(3):032318, November 2016.

[59] Raban Iten, Romain Moyard, Tony Metger, David Sutter, and Stefan Woerner. Exact and practical pattern matching for quantum circuit optimization. *ACM Transactions on Quantum Computing*, 3(1):1–41, January 2022.

[60] Raban Iten, Oliver Reardon-Smith, Emanuel Malvetti, Luca Mondada, Gabrielle Pauvert, Ethan Redmond, Ravjot Singh Kohli, and Roger Colbeck. Introduction to UniversalQCompiler, 2019.

[61] Siddharth Jain. Solving the Traveling Salesman Problem on the D-Wave quantum computer. *Frontiers in Physics*, 9:760783, November 2021.

[62] Sabine Jansen, Mary-Beth Ruskai, and Ruedi Seiler. Bounds for the adiabatic approximation with applications to quantum computation. *Journal of Mathematical Physics*, 48(10):102111, 2007.

[63] A.W. Johnson and S.H. Jacobson. On the convergence of generalized hill climbing algorithms. *Discrete Applied Mathematics*, 119(1-2):37–57, June 2002.

[64] Tadashi Kadowaki and Hidetoshi Nishimori. Quantum Annealing in the transverse Ising model. *Physical Review E*, 58:5355–5363, November 1998. DOI: 10.1103/PhysRevE.58.5355.

[65] George Karypis and Vipin Kumar. METIS – unstructured Graph Partitioning and Sparse Matrix ordering system, Version 2.0. *Technical report*, January 1995.

[66] B. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, February 1970.

[67] Alexey Yu. Kitaev. Quantum computations: algorithms and error correction. *Russian Mathematical Surveys*, 52(6):1191–1249, December 1997.

[68] kiutra GmbH. Quantum computer temperature: Do they need to be cold?, 2023.

[69] D.E. Knuth. *The Art of Computer Programming*, volume 4A of *Addison-Wesley series in computer science and information processing*. Addison-Wesley Publishing Company, Boston, MA, 2011.

[70] Abhoy Kole, Kamalika Datta, and Indranil Sengupta. A new heuristic for N-Dimensional Nearest Neighbor realization of a quantum circuit. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37:182–192, 2018.

[71] Cheng-Wen Lee and Bing-Yi Lin. Application of hybrid quantum tabu search with Support Vector Regression (SVR) for load forecasting. *Energies*, 9(11):873, October 2016.

[72] Menghan Li, Huanqing Cui, Chuanai Zhou, and Shaohua Xu. Gap: Genetic algorithm based large-scale Graph Partition in heterogeneous cluster. *IEEE Access*, PP:1–1, August 2020.

[73] Shang-Wei Lin, Tzu-Fan Wang, Yean-Ru Chen, Zhe Hou, David Sanán, and Yon Shin Teo. A parallel and distributed quantum SAT solver based on entanglement and quantum teleportation, 2023.

[74] Yuan-Ting Liu, Kai Wang, Yuan-Dong Liu, and Dong-Sheng Wang. A survey of universal quantum von Neumann architecture. *Entropy*, 25(8), 2023.

[75] Andrew Lucas. Ising formulations of many NP problems. *Frontiers in Physics*, 2, 2014.

[76] Martin Lukac, Marek Perkowski, Hilton Goi, Mikhail Pivtoraiko, Chung Hyo Yu, Kyusik Chung, Hyunkoo Jeech, Byung-Guk Kim, and Yong-Duk Kim. Evolutionary approach to quantum and reversible circuits synthesis. *Artificial Intelligence Review*, 20(3/4):361–417, December 2003.

[77] Frédéric Magniez, Ashwin Nayak, Peter C. Richter, and Miklos Santha. On the hitting times of quantum versus random walks. *Algorithmica*, 63(1-2):91–116, June 2012.

[78] Mariia Maltseva, Enrico Blanzieri, and Alexander Rumyantsev. Quantum circuit optimization via Graph Partitioning by Quantum Annealing. *Lobachevskii Journal of Mathematics*, 45(10):5126–5140, 2024.

[79] Mariia Maltseva, Enrico Blanzieri, and Alexander Rumyantsev. Quantum circuit optimization via local qubit reordering by Quantum Annealing. *Vestnik Tomskogo gosudarstvennogo universiteta. Upravlenie, vychislitelnaja tehnika i informatika – Tomsk State University Journal of Control and Computer Science*, 69:103–111, 2024.

[80] D. Maslov. Reversible Logic Synthesis Benchmarks Page. `https://reversiblebenchmarks.github.io/`. [Online; accessed 18-January-2025].

[81] D. Maslov and G.W. Dueck. Reversible cascades with minimal garbage. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(11):1497–1509, November 2004.

[82] D. Maslov, G.W. Dueck, D.M. Miller, and C. Negrevergne. Quantum circuit simplification and level compaction. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(3):436–444, March 2008.

[83] Olivia Di Matteo and Michele Mosca. Parallelizing quantum circuit synthesis. *Quantum Science and Technology*, 1(1):015003, March 2016.

[84] Catherine McGeoch, Pau Farré, and Kelly Boothby. The D-Wave Advantage2 Prototype. *D-Wave Technical Report Series*, pages 14–1063A–A, 2022.

[85] Satoshi Morita and Hidetoshi Nishimori. Convergence theorems for Quantum Annealing. *Journal of Physics A: Mathematical and General*, 39(45):13903–13920, November 2006.

[86] Beatrice Nash, Vlad Gheorghiu, and Michele Mosca. Quantum circuit optimizations for nisq architectures. *Quantum Science and Technology*, 5(2):025010, March 2020.

[87] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[88] Masanori Ohya and Natsuki Masuda. NP problem in quantum algorithm. *Open Systems & Information Dynamics*, 7, 11 1998.

[89] Oumarou Oumarou, Alexandru Paler, and Robert Basmadjian. QUANTIFY: A Framework for Resource Analysis and Design Verification of Quantum Circuits. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 126–131, Limassol, Cyprus, July 2020. IEEE.

[90] Davide Pastorello and Enrico Blanzieri. Quantum Annealing learning search for solving QUBO problems. *Quantum Information Processing*, 18(10):303, 2019.

[91] Davide Pastorello, Enrico Blanzieri, and Valter Cavecchia. Learning adiabatic quantum algorithms over optimization problems. *Quantum Machine Intelligence*, 3(1):2, June 2021.

[92] Siddhartha Patra, Saeed S. Jahromi, Sukhbinder Singh, and Román Orús. Efficient tensor network simulation of IBM's largest quantum processors. *Physical Review Research*, 6:013326, March 2024.

[93] Elijah Pelofske. 4-Clique network minor embedding for quantum annealers. *Physical Review Applied*, 21(3), March 2024.

[94] Marek Perkowski, Martin Lukac, Dipal Shah, and Michitaka Kameyama. Synthesis of quantum circuits in linear Nearest Neighbor model using Positive Davio Lattices. *Facta universitatis - series: Electronics and Energetics*, 24, January 2011.

[95] Frank Phillipson. Quantum computing in logistics and supply chain management an overview, 2024.

[96] John Preskill. Quantum computing in the NISQ era and beyond. *Quantum*, 2:79, August 2018.

[97] Mehdi Saeedi, Robert Wille, and Rolf Drechsler. Synthesis of quantum circuits for linear Nearest Neighbor architectures. *Quantum Information Processing*, 10(3):355–377, June 2011.

[98] R. A. M. Santos and R. Portugal. Quantum hitting time on the complete graph. *arXiv:0912.1217 [quant-ph]*, December 2009. arXiv: 0912.1217.

[99] Maximilian Schlosshauer. Quantum decoherence. *Physics Reports*, 831:1–57, October 2019.

[100] Michal Sedlák and Martin Plesch. Towards optimization of quantum circuits. *Open Physics*, 6(1), January 2008.

[101] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Optimization of quantum circuits for interaction distance in linear Nearest Neighbor architectures. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2013.

[102] Alireza Shafaei, Mehdi Saeedi, and Massoud Pedram. Qubit placement to minimize communication overhead in 2D quantum architectures. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pages 495–500, January 2014.

[103] Vivek V. Shende and Igor L. Markov. On the CNOT-cost of Toffoli gates. *Quantum Information & Computation*, 9(5):461–486, May 2009. Place: Paramus, NJ Publisher: Rinton Press, Incorporated.

[104] Vivek V. Shende, Igor L. Markov, and Stephen S. Bullock. Minimal universal two-qubit controlled-NOT-based circuits. *Physical Review A*, 69(6):062321, June 2004.

[105] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, 1994.

[106] Peter W. Shor. Scheme for reducing decoherence in quantum computer memory. *Physical Review A*, 52:R2493–R2496, October 1995.

[107] Tycho Sleator and Harald Weinfurter. Realizable universal quantum logic gates. *Physical Review Letters*, 74:4087–4090, May 1995.

[108] Mateusz Slysz, Krzysztof Kurowski, and Grzegorz Waligóra. Solving combinatorial optimization problems on a photonic quantum computer, 2024.

[109] Juexiao Su. Towards quantum computing: Solving satisfiability problem by Quantum Annealing. *UCLA Electronic Theses and Dissertations*, 2018.

[110] Ryan Sweke, Frederik Wilde, Johannes Meyer, Maria Schuld, Paul K. Faehrmann, Barthélémy Meynard-Piganeau, and Jens Eisert. Stochastic gradient descent for hybrid quantum-classical optimization. *Quantum*, 4:314, August 2020. arXiv: 1910.01155.

[111] Márcio M. Taddei, Jaime Cariñe, Daniel Martínez, Tania García, Nayda Guerrero, Alastair A. Abbott, Mateus Araújo, Cyril Branciard, Esteban S. Gómez, Stephen P. Walborn, Leandro Aolita, and Gustavo Lima. Computational advantage from the quantum superposition of multiple temporal orders of photonic gates. *PRX Quantum*, 2:010320, February 2021.

[112] Emiliano Tolotti, Enrico Zardini, Enrico Blanzieri, and Davide Pastorello. Ensembles of Quantum Classifiers. *Quantum Information & Computation*, 24(3&4):181–209, March 2024. arXiv:2311.09750 [cs].

[113] Hayato Ushijima-Mwesigwa, Christian F. A. Negre, and Susan M. Mniszewski. Graph Partitioning using Quantum Annealing on the D-Wave system. In *Proceedings of the Second International Workshop on Post Moores Era Supercomputing*, PMES'17, pages 22–29, New York, NY, USA, 2017. Association for Computing Machinery.

[114] Hattori Wakaki and Shigeru Yamashita. Mapping a quantum circuit to 2D Nearest Neighbor architecture by changing the gate order. *IEICE Transactions on Information and Systems*, E102.D:2127–2134, November 2019.

[115] Dong-Sheng Wang. A comparative study of universal quantum computing models: Toward a physical unification. *Quantum Engineering*, 3(4), December 2021.

[116] Jonathan Welch, Alex Bocharov, and Krysta M. Svore. Efficient approximation of diagonal unitaries over the Clifford+T basis. *Quantum Information & Computation*, 16(1–2):87–104, Januray 2016. Place: Paramus, NJ Publisher: Rinton Press, Incorporated.

[117] Robert Wille, Daniel Große, Lisa Teuber, Gerhard W. Dueck, and Rolf Drechsler. Revlib: An online resource for reversible functions and reversible circuits. In *38th International Symposium on Multiple Valued Logic (ismvl 2008)*, pages 220–225, 2008.

[118] Robert Wille, Oliver Keszocze, Marcel Walter, Patrick Rohrs, Anupam Chattopadhyay, and Rolf Drechsler. Look-ahead schemes for Nearest Neighbor optimization of 1D and 2D quantum circuits. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 292–297, Macao, Macao, January 2016. IEEE.

[119] Robert Wille, Aaron Lye, and Rolf Drechsler. Optimal SWAP gate insertion for Nearest Neighbor quantum circuits. In *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 489–494, Singapore, January 2014. IEEE.

[120] Robert Wille, Mehdi Saeedi, and Rolf Drechsler. Synthesis of reversible functions beyond gate count and quantum cost, April 2010. arXiv:1004.4609 [quant-ph].

[121] Dennis Willsch, Madita Willsch, Carlos D. Gonzalez Calaza, Fengping Jin, Hans De Raedt, Marika Svensson, and Kristel Michielsen. Benchmarking Advantage and D-Wave 2000Q quantum annealers with exact cover problems. *Quantum Information Processing*, 21(4), April 2022.

[122] Xin-Chuan Wu, Marc Grau Davis, Frederic T. Chong, and Costin Iancu. QGo: Scalable quantum circuit optimization using automated synthesis, March 2022. arXiv:2012.09835 [quant-ph].

[123] Yao Xiao, Shahin Nazarian, and Paul Bogdan. A stochastic quantum program synthesis framework based on Bayesian optimization. *Scientific Reports*, 11(1):13138, December 2021.

[124] Sheir Yarkoni, Elena Raponi, Thomas Bäck, and Sebastian Schmitt. Quantum Annealing for industry applications: introduction and review. *Reports on Progress in Physics*, 85(10):104001, September 2022.

[125] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. QFAST: Quantum synthesis using a hierarchical continuous circuit space. *arXiv:2003.04462 [quant-ph]*, March 2020.

[126] Ed Younis, Koushik Sen, Katherine Yelick, and Costin Iancu. QFAST: Conflating search and numerical optimization for scalable quantum circuit synthesis. *arXiv:2103.07093 [quant-ph]*, December 2021.

[127] Douglas Youvan. Quantum computation at absolute zero: Theoretical implications and potential. June 2024.

[128] Yingchen Yu. A research review on job-shop scheduling problem. *E3S Web of Conferences*, 253:02024, January 2021.

[129] Alwin Zulehner, Stefan Gasser, and Robert Wille. Exact global reordering for Nearest Neighbor quantum circuits using A^*. In Iain Phillips and Hafizur Rahaman, editors, *Reversible Computation*, volume 10301, pages 185–201. Springer International Publishing, Cham, 2017. Series Title: Lecture Notes in Computer Science.

[130] Alwin Zulehner, Alexandru Paler, and Robert Wille. An efficient methodology for mapping quantum circuits to the IBM QX Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(7):1226–1236, 2019.