The School of Mathematics

Quantum Annealing Algorithms for Scientific Computing and Machine Learning

by

Changliang Wei

Dissertation Presented for the Degree of
MSc in Computational Applied Mathematics

August 2023

Supervised by
Dr Matias Ruiz

Abstract

Combinatorial optimization constitutes a pivotal research area of applied mathematics, with widespread applications across several scientific disciplines. However, when dealing with large-scale optimization problems, many numerical methods are hard to escape from the local minima. The quantum annealing algorithm is based on the quantum properties to find the global minimum of the combinatorial optimization problems. However, implementing the quantum annealing algorithm on a real quantum computer has many restrictions, including difficulty dealing with noise, having limited qubit count, difficulty encoding complex problems, and many others. We employ quantum-inspired, classical algorithms to explore the potential of harnessing quantum properties for solving combinatorial optimization problems. In particular, we implement the so-called Simulated Quantum Annealing (SQA) algorithm to tackle problems in scientific computing and machine learning. We first discuss how to write and solve linear systems of equations as a binary combinatorial optimization problem. Second, we investigate using SQA for the pruning of neural networks. Our findings show that increasing the quantum behaviour of the algorithm yields better results, thus foreshadowing the possibility of achieving enhanced performance when the algorithm is eventually deployed on an actual quantum computing platform.

Acknowledgments

Own Work Declaration

I declare that, unless otherwise stated, the following work is my own.

Contents

List of Tables

List of Figures

1 Introduction

1.1 Motivation

In real-life scenarios, optimization problems manifest in many different ways, such as chemical simulations [45], industrial environments [56], structural optimization [31], and many others. In mathematics, the essence of an optimization problem lies in determining an optimal solution that minimizes or maximizes a provided cost function, often referred to as an objective function. Based on the presence of constraints on the variables, the optimization problem can be categorized into two types: constrained optimization [4] and unconstrained optimization [10]. In constrained optimization, variables are restricted to specific values within a defined range, while in unconstrained optimization, variables can assume any value. Furthermore, optimization problems can also be categorized based on the nature of variables: discrete or combinatorial optimization (CO) [30], involving discrete variables, and continuous optimization, involving continuous variables.

Numerous methods have been proposed to solve optimization problems, with specific approaches for individual problem types. The brute force approach is commonly employed to solve discrete optimization problems. This method involves the analysis of all possible variables, ensuring to find the optimal solution. However, the computational complexity grows exponentially with the problem size, making it unsuitable for solving large-scale optimization problems. Optimization solvers can be categorized into deterministic and stochastic methods. Deterministic exploration methods can be classified into three categories: second-order methods (e.g. the Newton method [15]), which solving use the gradient and Hessian matrices; first-order methods (e.g. the quasi-Newton method [58]), which solving use the gradient; and zero-order methods (e.g. the Pattern Searched method [2]), relying only on the objective function itself for solving optimization problems. Deterministic exploration methods provide a theoretical guarantee of local optima. However, they are prone to becoming trapped in local minima. Over the past two decades, a lot of stochastic and heuristic approaches have been proposed, including Particle Swarm Optimization (PSO) [47], Differential Evolution (DE) [50], Harmony Search (HS) [16], Random Perturbation methods (RP) [57], Simulated Annealing (SA) [28], and many others. Numerous heuristics are inspired by physical phenomena, as seen in methods like simulated annealing (SA). However, these methods often require numerous steps to ensure convergence, leading to unreasonable computation times.

Over the past two decades, there has been a growing interest in using quantum computing to solve optimization problems, aiming to circumvent the limitations of classical methods. Using various quantum properties, such as superposition, entanglement, and tunnelling, can aid in developing less computationally complex algorithms, such as quantum annealing (QA). The optimization problems using QA are also known as quantum adiabatic optimization [14][25]. Due to their exponential speed-ups compared to classical algorithms, quantum computing algorithms have the potential to revolutionize various realms of research and engineering. However, it will likely take decades before practical implementation of quantum computers can provide exponential speed-ups. During the wait for improved hardware, a category of heuristic quantum algorithms, known as near-term quantum algorithms—examples include simulated quantum annealing (SQA)—are being developed to harvest quantum speed-ups. Theoretical work by Crosson and Harrow [7] proved that SQA achieves exponential speed-up over SA for specific problems, such as optimizing spike cost functions. Research has demonstrated that the advantages of adiabatic evolution tunnelling across energy barriers can also be realized in classical optimization algorithms.

The Quadratic Unconstrained Binary Optimization (QUBO) problem and the Ising model are the most feasible formulations for solving combinatorial optimization (CO) problems using a quantum annealer. Hence, this thesis is motivated by exploring the SQA algorithm based on the Path Integral Monte Carlo (PIMC) method to solve the QUBO problem and the Ising model. Solving linear systems is a fundamental problem across various scientific disciplines. This problem can be reformulated as a QUBO problem and solved using the SQA algorithm, as discussed in Section 3. However, being a relatively straightforward problem, numerous numerical methods have been proposed for solving linear systems. Therefore, this thesis will provide only a concise overview in this section. This problem is solely utilized to demonstrate the effectiveness of the SQA algorithm in solving the QUBO problem. The neural network structure optimization is a large-scale problem and is the category of NP-hard problems. One of the neural network structure optimizations is pruning, using reduced connective of the neural network to compress the neural network. A binary variable can represent an edge within the neural network. Here, '1' means the edge has remained, and '0' means the edge has pruned. Consequently, the problem of neural network pruning can be formulated as a Binary Optimization (BO) problem. Kuo et al. [32] proposed using a classical algorithm, the SA, by customizing its components to solve this problem. With the increasing size of neural networks, solving this problem using classical algorithms becomes relatively inexpensive. Quantum computing offers remarkable computational power, inspiring us to employ quantum algorithms, especially QA, to solve this problem. However, due to factors such as the network's depth and the nonlinearity of its activation functions, this problem proves challenging to reformulate as a QUBO problem. Theoretically, a quantum annealer is hard to solve the neural network pruning problem. SQA is a classical algorithm to simulate the properties of quantum, which do not have this restriction. Hence, the SQA-based pruning method is proposed to solve the neural network pruning problem in the classical computer while employing experiments to investigate the efficacy of this approach. A large part of this thesis will be devoted to exploring this aspect.

1.2 Outline of Thesis

The thesis is organized as follows. Section 2 reports the background and basics knowledge, including the QUBO and Ising formulation in Section 2.1, the heuristic search method and simulated annealing algorithm in Section 2.2, some quantum stuff in Section 2.3 including adiabatic quantum computing and quantum annealing, and the SQA algorithm based on the Path Integral Monte Carlo (PIMC) method in Section 2.4. The SQA algorithm for solving linear systems consists of Section 3, which starts from simple binary variables in Section 3.1. Then it extends to floating-point calculation in Section 3.2. Section 4 constructs the SQA-based pruning method for solving the neural network structure optimization problem. The description of neural network pruning is shown in Section 4.1. Section 4.2 reports the details of the SQA-based pruning steps and implementation in Section 4.3. Finally, in Section 5, conclusions are drawn, and future perspectives are illustrated.

2 Background

2.1 Optimization Problems Formulation

Optimization problems can be categorized into continuous and discrete classes based on the nature of their variables. Conventionally, the optimization problems with discrete variables are also called combinatorial optimization (CO) problems [36]. The Quadratic Unconstrained Binary Optimization (QUBO) problem and the Ising model are the most viable formulations for solving combinatorial optimization (CO) using quantum algorithms. This section will present the formulations of the QUBO problem and the Ising model.

2.1.1 QUBO Formulation

QUBO is a combinatorial optimization problem widely described in many real-world problems [18], such as allocation [12], placement problems [49], nurse scheduling problem [24], vehicle routing problem [51], and many others. Like its name, the QUBO problem is defined on binary variables $x_i \in \{0, 1\}$, and the highest order term of its objective function is quadratic. Unconstrained means that the objective function should be minimized or maximized without any constraints on defining the variables in the problem. The objective function of the QUBO problem in mathematics can be defined as

$$f(\mathbf{x}) = c + \sum_i b_i x_i + \sum_{(i,j)} a_{ij} x_i x_j. \tag{2.1}$$

Then then QUBO problem here can be defined in matrix form as

$$\text{QUBO : minimize/maximize} \quad f(\mathbf{x}) = \mathbf{x}^\intercal A \mathbf{x}, \tag{2.2}$$

where $\mathbf{x}$ is defined as binary variables, and A is a square matrix of constraints, which would be designed as the specific problem. It is common to assume that the matrix A is symmetric or in the upper triangular form in some problems [18]. To symmetric form, the element a_{ij} could be replace by $(a_{ij} + a_{ji})/2$. The matrix A can be in upper triangular form if the third term in Eq.(2.1) is rewritten as $\sum_{i<j} a_{ij} x_i x_j$, then a_{ij} could be replaced by 0 as $i > j$.

The general QUBO problem can also start with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the vertex set and $\mathcal{E}$ is the edge set, and the fully connected graph K_5 is shown in Figure 1. The QUBO Hamiltonian on G is defined as

$$H(\mathbf{x}) = -\sum_{(i,j)} J_{ij} x_i x_j - \sum_i h_i x_i, \tag{2.3}$$

where $x_i \in \{0, 1\}$ for all $i \in V$. The coefficient J_{ij} is called the strength between vertices i and j, and the coefficient h_i is called the weight at vertex i. Eq.(2.3) is also called the objective function in the optimization problem.

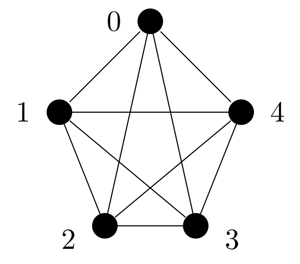


Figure 1: The fully connected graph K_5 for the QUBO problem, the vertex set $\mathcal{V}_5 = \{0, 1, 2, 3, 4\}$ and the edge set $\mathcal{E}_4 = \{\{0, 1\}, \{0, 2\}, \{0, 3\}, \{0, 4\}, \{1, 2\}, \{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$.

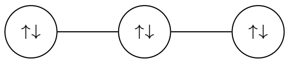
2.1.2 Ising Model Formulation

A physical-mathematical model called the Ising model is used to investigate phase transitions and other features of physical systems that change over time. The problem involves a set of n atomics (or particles) arranged on the edges of a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Each atomic can be shown in one of two spin states, represented by ± 1. The state 1 means the atomic spin upward, and the state -1 means the spin downward. A spin configuration $\sigma = \sigma_1 \ldots \sigma_n$ assigns spin values of all atomic states. The spin state of each atom only interacts with its neighbours. The Hamiltonian of an Ising model is defined as follows,

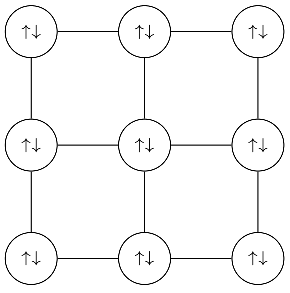
$$H(\sigma) = -\sum_{(i,j)} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i, \tag{2.4}$$

where J_{ij} is called interaction term, and h_i is called the external magnetic field coefficient of the atomic i. The interaction term J_{ij} controls the weight of the spin coupling of neighbour atoms (i, j), and its sign indicates whether the neighbour atoms have the same spin orientation preference. The external term h_i control the weight of the atom itself spin. Here, the Hamiltonian of the Ising model in Eq.(2.4) can be related to the Hamiltonian of the QUBO problem under a simple change. Hence, they can often be viewed as equivalence problems [38]. By two simple mappings, $x_i = (1 + \sigma_i)/2$, and $\sigma_i = 2x_i - 1$, they could be moved from one formulation to another.

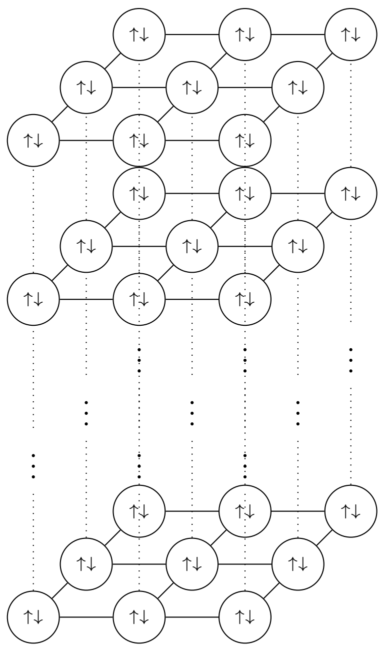
Depending on each atom having different neighbours, the Ising model has different structures, shown in Figure 2. In the $1D$ Ising model, each atom has two neighbours (Figure 2a); in the $2D$ Ising model, each atom has four neighbours (Figure 2b). Each atom has six neighbours in the $3D$ Ising model (Figure 2c). Except for these three structures, the Ising model has fully connected structures, the same as the fully connected graph for the QUBO problem (Figure 1). CO problems can be shown as minimising or maximising the Hamiltonian of QUBO (Eq.(2.3)) or Ising model (Eq.(2.4)) to find the optimal spin configurations.



(a) $1D$ Ising Structure



(b) $2D$ Ising Structure



(c) $3D$ Ising Structure

Figure 2: The Ising model structures of $1D$, $2D$, and $3D$. Each atom has two spin state, upward or downward.

2.2 Classical Annealing Algorithm

In the previous section, two optimization problems were introduced (perfectly equivalent). In this section, one type of classical annealing algorithm will be introduced to solve the combinatorial optimization problems, i.e. Simulated Annealing (SA). Simulated Annealing (SA) is one of the heuristic search methods. Hence, the definition of heuristic search methods will be introduced first.

2.2.1 Heuristic Search Methods

In mathematical programming, heuristic algorithms are proposed to find near-optimal solutions for optimization problems within a certain amount of time (or iterations). However, this is achieved by sacrificing optimality, completeness, accuracy, and precision for speed. Heuristics are problem-specific [28]; for different types of optimization problems, specific heuristics methods are used. In other words, the heuristics method will be designed to solve a specific optimization problem and should be redesigned for other problems. Heuristic search is an algorithm that finds the best solution to a problem by heuristics, using some rules that can be used to make a decision or solve a problem. Here, the most common definition of the heuristic search method is presented [37].

Suppose that a minimize optimization problem like the QUBO problem shown in Eq.(2.3), defined on binary variables $\mathbf{x} = x_1x_2\dots x_n$ The objective function $f(\mathbf{x})$ is calculated a cost for each feasible solution $\mathbf{x} \in \mathbb{B} = \{0,1\}^n$. The minimize optimization problem is to find the optimal solution which has the minimize value of the objective function.

The heuristic search method has two critical components, solution space and local search algorithm. For different specific problems, the heuristic search method requires to design these two components. The solution space $\mathbb{D}^n$ depends on the feasible solutions. In order to be more intuitive present, the solution space can be shown as a graph. The neighbourhood rule defines the left or right sides of one feasible solution. The different decisions of neighbourhood rule will affect the effectiveness of the heuristic search method in specific ways. The graph and the objective function create a landscape where hilltops correspond to solutions with high-cost and valleys to low-cost solutions, shown in Figure 3. The local search algorithm can use specific rules to move from low-cost to high-cost places to escape from local minima. The local search algorithm begins at an initial state in the solution space. Then using the neighbourhood rule iterates from node to neighbour node and generally toward the low-cost place in Figure 3, until terminated by some stopping rule.

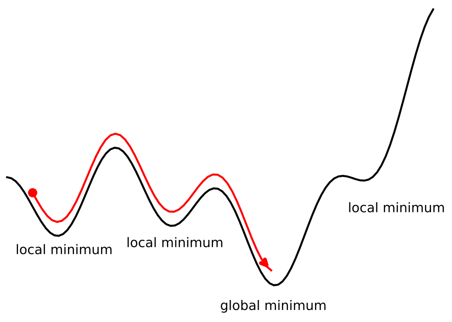


Figure 3: The solution landscape is established by the neighborhood rule and objective function. The heuristic search method avoids hitting local minima by moving from one node in the local minimum to neighbor node.

2.2.2 Simulated Annealing

The simulated annealing (SA), proposed by Kirkpatrick et al. [28], is a heuristic search method for solving combinatorial optimization problems. The SA algorithm is inspired by annealing in solid physics. This process involves a solid entering a low-energy state after increasing its temperature. The main advantage of SA is its simplicity [9]. Hence it is heavily used in real life. SA is based on two steps of heating and cooling, the heating step moves the atoms in the lattice, and the cooling step finds the optimal configuration of the atoms (optimal solution). The results[28] show that the minimum energy state can be reached as long as the initial temperature is high enough and the cooling time is long enough. The SA algorithm allows atoms to avoid locally optimal configurations.

The details of the SA shows in Algorithm 1. The SA algorithm has four main steps: (1) initialization, (2) generating a new solution, (3) accepted or rejected, and (4) cooling scheme.

The first step is to initialize the parameters. The SA algorithm needs two initialisation parameters: initial solution x and initial temperature T. The initial solution x is generated from the solution space. The solution space is defined as the specific optimization problem. The method of generating the initial solution is depended on specific demand. The usual way is using random. The initial temperature T is stated at a high value to simulate the process of solid heating. The typical approach to initialize temperature T is choice a value near the $E[|f(x) - f(x_{\text{new}})|]$, which shows the mean value of the difference of the objective function of the initial solution x with its all neighbours. In experiments, a more straightforward approach could be used using random samples of the solution space and using the $|f_{\max} - f_{\min}|$ to as the initialize temperature T.

The second step of the SA algorithm is to generate a new solution using the neighbourhood rule. The neighbourhood rule is designed by the specific problem, which decides the random walk that can be made. The neighbourhood rule's design will affect the SA algorithm's convergence. A reasonable neighbourhood rule design can help the SA algorithm's process to converge quickly. For an example of a neighbourhood rule in one QUBO problem, the initial solution $x = [0, 0, 1]$, one of the neighbours of x can be $x_{\text{new}} = [0, 0, 0]$. The total number of the neighbour of x is three.

The third step of the SA algorithm is to accept or reject this choice of neighbour. In the SA algorithm, this step to decide whether to accept or reject the new solution (the step of random walk) uses the Metropolis-Hastings (MH) algorithm. The MH algorithm is a popular Markov chain Monte Carlo method for obtaining complex sequences of random samples [53]. When the new solution $x_{\text{new}}$is generated using the neighbourhood rule, if $f(x_{\text{new}}) < f(x)$, the new solution x_{new}(or this step of random walk) will be accepted. If $f(x_{\text{new}}) \geqslant f(x)$, the new solution x_{new}(or this step of random walk) will be accepted using the accepted rule in the local search method as mentioned above will be rejected. However, the new solution x_{new} will not be rejected directly in the MH algorithm. This uphill move will be a certain probability to be accepted using the Boltzmann distribution. The acceptance principle using the MH algorithm is defined as follows.

Definition 2.1 *[9] Suppose that the minimize optimization problem P, the objective function $f(x)$, the current solution x and the new solution x_{new} using the neighbourhood rule of x. The acceptance probability of this step of the random walk is defined as*

$$AcceptProb(T, \Delta) = \min\left(1, e^{-\Delta/k_B T}\right), \tag{2.5}$$

where $\Delta = f(x_{new}) - f(x)$, T is the current temperature, and k_B is the Boltzmann constant.

MH algorithm allows the random walk to move to the higher value of the objective function at a high temperature T in Figure 3, which lets the simulated annealing can escape from the local minimum.

The last step is the cooling schedule of the temperature. This step has simulated the annealing in solid physics. The temperature T is initialized at a high value and then decremented at each iteration. One typical cooling schedule is according to a monotonically decreasing function

$T(i) = a/\ln(i)$ for iteration i and a constant a, stopping when T is close to zero or meeting the max iteration steps. This cooling schedule is known as the logarithmic cooling.

Theorem 2.1 *[37] If the cooling schedule of the temperature T is*

$$T(i) \geqslant cn/\ln(i), \tag{2.6}$$

for constant c, and iteration i. The SA is guaranteed to find the optimal solution for $i \to \infty$.

However, the logarithmic cooling schedule requires slow cooling enough. Instead, another common approach [32] is using geometric cooling,

$$T(i+1) = \eta T(i). \tag{2.7}$$

Here, the cooling rate η is used to control the temperature decrease. If the temperature T is high, $e^{-\Delta/T}$ would be close to 1. The uphill moves (or worse) have a high probability of being accepted. As the temperature is reduced increasingly, the uphill move is more likely to be rejected.

2.3 Quantum Computing

In 1994, Shor et al. [48] proposed a prime factorization algorithm based on quantum computing. Solving this problem could take billions of years on a classical computer. However, theoretically, it should only take a few hours in a quantum computer [41]. This discovery sparked researchers' interest in quantum computing and accelerated the development of quantum computing technology. The difference between classical and quantum computers is how the information is shown. In the classical computer, the information is shown as a register R of bits, which can be represented as '0' and '1'. In quantum computers, the information is represented by the quantum bits (or qubits), which also can be shown as '0' and '1' in simulation. However, the most critical property of qubit is superposition which can be combined with other configurations. This property allows qubits can store more information than bits [29]. Another critical property of qubits is entanglement. Preskill [40] shows an example using the book to describe this quantum phenomenon. Suppose that a quantum book, the context of this book is entanglement. It is hard to understand the book if the way to read it is to read it page by page. The correct method to read this book is to observe more pages simultaneously. This order the quantum computing is very different from classical computing.

2.3.1 Adiabatic Quantum Computing

Farhi et al. [13][14] proposed the Adiabatic Quantum Computing (AQC) in 2000. These AQC algorithms are one new quantum computing class based on the adiabatic evolution [14]. The AQC algorithms show that by controlling the evolution of the quantum system slowly enough, then the quantum system can stay near its ground state.

The AQC algorithms are defined to solve optimization problems using quantum properties. The objective function of the AQC algorithms is described by the time-varying Hamiltonian $\mathcal{H}(t)$, which is shown as follows:

$$\mathcal{H}(t) = s(t)\mathcal{H}_I + (1 - s(t))\mathcal{H}_F, \tag{2.8}$$

where $\mathcal{H}_I$ is the initial Hamiltonian, which is chosen as the quantum system's initial state; $\mathcal{H}_F$ is the finial Hamiltonian, which is shown as the objective function of the optimization problems to find the minimum state; and $s(t)$ is called the adiabatic evolution path, which is decreased from 1 to 0 as t increasing. Hence, at the initial time $t = 0$, the Hamiltonian of the quantum system is at initial state $\mathcal{H}(0) = \mathcal{H}_I$. Through the t increase to the end of the evolution t_f, the Hamiltonian of the quantum system is to find the ground state, which is the final Hamiltonian $\mathcal{H}(t_f) = \mathcal{H}_F$. The simple adiabatic evolution path is a linear function, shown as $s(t) = 1 - t/t_f$.

The AQC algorithms are based on the adiabatic theorem to design for solving optimization problems. Born and Fock proposed the adiabatic theorem [5] in 1928. Many researchers gave discussions and proofs, like Aharonov et al. [1], Reichardt [42], Van Dam [52], and many others. Here the adiabatic theorem is briefly given as follows.

Theorem 2.2 *[26][37] The adiabatic theorem was formulted to describe certain properties of quantum particle processes, which evolve according to the Schrödinger equation,*

$$i\hbar\frac{d}{dt}|\Phi_t\rangle = \mathcal{H}(t)|\Phi_t\rangle.$$

Here i is the imaginary unit and $\hbar = h/2\pi$ where h is Planck's constant. The original system reaches thermal equilibruim (the ground state of the Schrödinger equation) in the long-time limit is satisfied if the following relation is satisfied:

$$\lim_{t\to\infty} \frac{\|\frac{d\mathcal{H}(t)}{dt}\|}{\Delta(t)^2} \ll 1.$$

2.3.2 Quantum Annealing

Quantum annealing (QA) is a class of quantum algorithms which use quantum properties to search for the optimal solution of the optimization problems. Instead of SA using the temperature T decrease to control the annealing process, QA is used the parameter Γ, which is called the transverse field coefficient (or called the tunnelling coefficient). In the SA algorithm, the thermal fluctuation phenomenon lets the system move from the local minimum state to other states, shown as the red line in Figure 4. In the QA algorithm, the quantum fluctuation phenomenon (quantum tunnelling) is used for state transition, which is shown as the blue line in Figure 4.

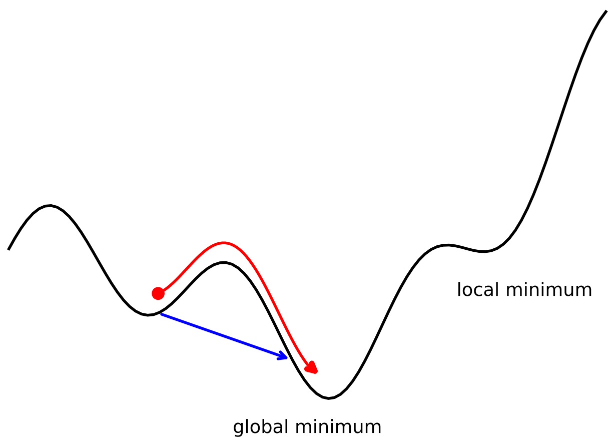


Figure 4: The solution landscape is established by the neighborhood rule and objective function. The red line is shown as the thermal fluctuation for SA, and the blue line is shown as the quantum fluctuation (tunnelling) for QA.

The objective function $f(x)$ in the QA algorithm can be represented by the Hamiltonian $\mathcal{H}(t)$, the same as adiabatic quantum computing in Eq.(2.8). The time-varying Hamiltonian of QA is shown as follows:

$$\mathcal{H}(t) = \mathcal{H}_F + \Gamma(t)\mathcal{H}_D, \tag{2.9}$$

where $\mathcal{H}_F$ is the finial Hamiltonian shown the ground state of the quantum system, $\mathcal{H}_D$ is called the transverse field Hamiltonian (or called the disordering Hamiltonian), and Γ is the transverse field which is initialized to a high value and gradually decrease to zero over the annealing process.

As mentioned aove, the most feasible formulations for solving optimization problems with quantum approaches are the QUBO and Ising problems, which introduced in Section 2.1. The Hamiltonian of the transverse-feild Ising model for QA is

$$\mathcal{H}(t) = \mathcal{H}_{\text{Ising}} + \Gamma(t)\mathcal{H}_{\text{TF}}, \tag{2.10}$$

where $\mathcal{H}_{\text{Ising}}$ denotes the Hamiltonian of the Ising model,

$$\mathcal{H}_{\text{Ising}} = -\sum_{(i,j)} J_{ij}\sigma_i^z\sigma_j^z - \sum_i h_i\sigma_i^z, \tag{2.11}$$

and $\mathcal{H}_{\text{TF}}$ is the Hamiltonian of the transverse field. The common choice of the Hamiltonian of

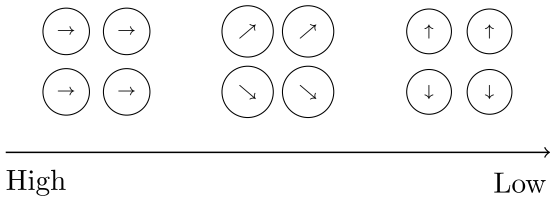


Figure 5: Spins configuration through the annealing process. From left to right, the value of a transverse field from high to low.

the transverse field is

$$\mathcal{H}_{\mathrm{TF}} = -\sum_i \sigma_i^x. \tag{2.12}$$

Here, σ_i^x and σ_i^z are the Pauli matrices, which shown as

$$\sigma^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \text{and} \quad \sigma^z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}.$$

At the beginning of the annealing process, the transverse field has a higher value, making the Hamiltonian of the transverse field dominant. The Eq.(2.12) is shown as all of the spins aligned on the x-axis with the same forward. Through the annealing process, the strength of the transverse field decreases, and the Hamiltonian of the Ising model is to be dominant. Eq.(2.10) shows that all of the spins gradually trend to the z-axis, and J and h make the spins go through the optimal configurations with the minimum value of Hamiltonian. The process of annealing is shown in Figure 5.

2.4 Simulated Quantum Annealing

Many possible ways to implement the classical QA algorithm are proposed to search the ground state of the time-dependent Hamiltonian in Eq.(2.9). This section focuses on simulated quantum annealing via Path Integral Monte Carlo (PIMC) [26], a type of Monte Carlo sampling method proposed by Battaglia et al [3].

SQA is a classical algorithm used to simulate some properties of QA. It has the same goal as QA: finding the Ising model's ground state or the optimal solution to the optimization problem. The transverse-field Ising model, which is shown in Eq.(2.10), could be using the Suzuki-Trotter formula to be rewritten as the corresponding Hamiltonian of the Ising model

$$\mathcal{H}(t) = -\sum_{k=1}^{M}\left(\sum_{(i,j)}\frac{J_{ij}}{M}\sigma_i^k\sigma_j^k + \sum_i \frac{h_i}{M}\sigma_i^k + \gamma(t)\sum_i \sigma_i^k\sigma_i^{k+1}\right), \tag{2.13}$$

where M is the total number of Trotter slices (or called replicas), σ_i^k is used to denote a classical Ising spin at the ith site on the kth replica, and the time-dependent coefficient of the term $\gamma(t)$ is given by

$$\gamma(t) = \frac{T}{2}\ln\left(\coth\left(\frac{\mathit{\Gamma}(t)}{MT}\right)\right). \tag{2.14}$$

Here $\gamma(t)$ represents the strength of couplings between the replicas. At the beginning of annealing, the coupling $\gamma(t)$ strength is high enough. Each replica will gradually have the same spin configuration. At the end of the annealing process, parameter $\gamma(t)$ will be decreased at small enough values close to zero. Eq.(2.13) is changed to the classical Ising model, which has the form

$$\mathcal{H}(t) = -\sum_{k=1}^{M}\left(\sum_{(i,j)}\frac{J_{ij}}{M}\sigma_i^k\sigma_j^k + \sum_i \frac{h_i}{M}\sigma_i^k\right), \tag{2.15}$$

With large enough M and small enough T, Eq. (2.13) converges to the thermal equilibrium of Eq. (2.10) and has a high probability of finding the ground state of the Ising model.

The details of SQA shows in Algorithm 2. The SQA algorithm has five main steps: (1) initialization, (2) generating a new solution, (3) accepted or rejected, (4) repeat steps (2) and (3) at each replica, (5) cooling scheme.

The first step is to initialize the parameters. The difference with the SA algorithm is that it not only initialises the temperature T, but SQA also needs to initialize two parameters: initial solution x and transverse field $\mathit{\Gamma}$. SQA needs to initialize M different solution x from the solution space for each replica. The method of generating the initial solution is depended on specific demand. The usual way is using random. The initialisation of the transverse field $\mathit{\Gamma}$ decided the convergence of the SQA algorithm and will be discussed later. The second and three steps of the SQA algorithm are similar to the SA algorithm to generate a new solution using the neighbourhood rule and decide to accept or reject. The fourth step is to repeat the second and three steps at each replica.

Like temperature T in SA, transverse field Γ starts at a high value and gradually decresed to zero according to a given schedule. Kimura et al. [26] deduce a condition for SQA to converge to a thermal equilibruim state by applying the adiabatic theorem, and derive a condition on the coefficient of the transverse field that ensures the convergence to a thermal equilibruim state. Consider a general function of the following form for the coefficient of the transverse field $\Gamma(t)$:

$$\Gamma(t) = MT\tanh^{-1}\left(\frac{1}{(4N)^{1/2N}(c_1t+c_2)^{1/2N}}\right), \tag{2.16}$$

where c_2 is an integral constant. With this form of $\Gamma(t)$ satisfies [26] the asymptotic adiabatic condition for the Schrödinger equation, given that c_1 is chosen to be small enough.

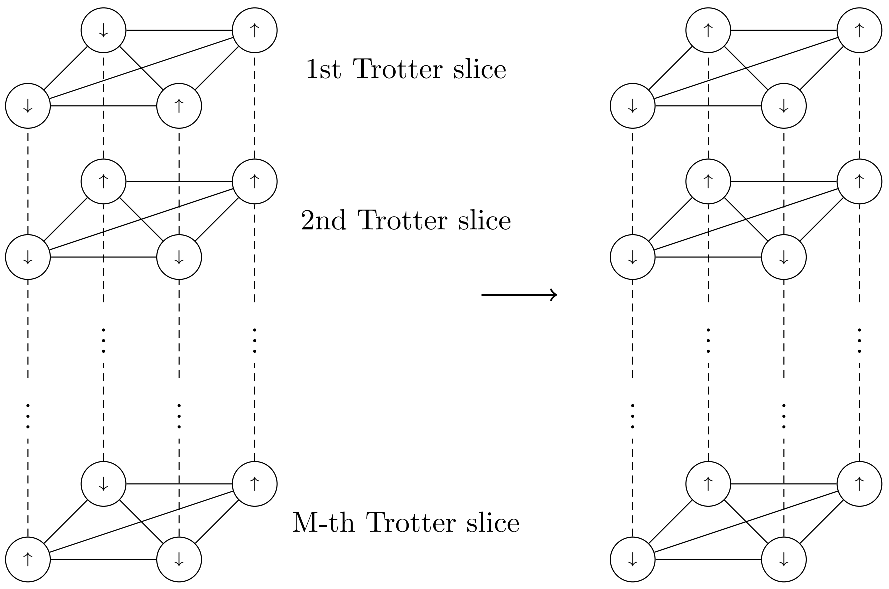


Figure 6: One-dimensional higher Ising model with M Trotter slices. The left plot shows the initial state of the annealing process, using a randomly chosen configuration for each slice, with up arrows showing $\sigma_i = 1$ and down arrows showing $\sigma_i = 1$. The plot on the right shows the optimal configuration of the Ising model, with the configuration having the lowest Hamiltonian magnitude at the end of the process.

3 Solving Linear Systems

Simulated quantum annealing algorithm based Path Integral Monte Carlo (PIMC) has introduced in Section 2.4. A famous example of the SQA algorithm is solving linear systems, a fundamental problem in many scientific disciplines. Unlike other numerical methods for solving linear systems, the linear systems need to be transformed into the QUBO problem first [44]. Then the SQA algorithm is used to solve the QUBO problem. In this section, the SQA algorithm for solving linear systems will be start from simple binary variables in Section 3.1. Then it extends to floating-point calculation in Section 3.2.

3.1 Binary-Point Calculation

Given a matrix $A \in \mathbb{R}^{n\times n}$, two vectors $\mathbf{x} \in \mathbb{R}^n$, and $\mathbf{y} \in \mathbb{R}^n$. The linear system can be defined as

$$A\mathbf{x} = \mathbf{y}. \tag{3.1}$$

Solving linear systems also is to solve the linear inverse problem, which given a matrix $A \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{y} \in \mathbb{R}^n$, to find the $\mathbf{x} \in \mathbb{R}^n$. The linear inverse problem can be defined as

$$\mathbf{x}_{LS} = \arg\min_{\mathbf{x}\in\mathbb{R}^n} \|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_2^2, \tag{3.2}$$

where λ is the parameter. If $\mathbf{x} \in \mathbb{B}^n$, we can rewrite the linear inverse problem with binary variables as a QUBO problem,

$$\begin{aligned} f(\mathbf{x}) &= \|A\mathbf{x}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{x}\|_2^2 \\ &= (\mathbf{x}^\intercal A^\intercal - \mathbf{y}^\intercal)(A\mathbf{x}-\mathbf{y}) + \lambda \mathbf{x}^\intercal \mathbf{x} \\ &= \mathbf{x}^\intercal (A^\intercal A + \lambda I)\mathbf{x} - 2\mathbf{y}^\intercal A\mathbf{x} + \mathbf{y}^\intercal \mathbf{y} \\ &= \sum_{(i,j)} \hat{A}_{ij} x_i x_j - 2\sum_{(i,j)} y_j A_{ji} x_i + \sum_i y_i^2, \end{aligned} \tag{3.3}$$

where $\hat{A} = A^\intercal A + \lambda I$, and $x_i \in \{0,1\}$. Since $\sum_{i=0}^{N-1} y_i^2 \geqslant 0$, remove this term will not affect the result for solving linear inverse problem. Hence, we can rewrite as QUBO Hamiltonian as (2.3),

$$\mathcal{H}(\mathbf{q}) = -\sum_{(i,j)} J_{ij} q_i q_j - \sum_i h_i q_i, \tag{3.4}$$

where $q_i = (1 + x_i)/2$, $J_{ij} = -\hat{A}_{ij}$, and $h_i = 2y_j A_{ji}$.

Example 1 *Given a simple linear system with binary variables,*

$$\begin{pmatrix} 5 & -7 \\ 3 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 12 \\ 5 \end{pmatrix}, \tag{3.5}$$

where $\mathbf{x} = [1, -1]^\intercal$.

Since the QUBO problem and the Ising model can be easily transformed into each other by mapping, then binary variables $\mathbf{x}$ here are written as $[1, -1]^\intercal$ which same as $[1, 0]^\intercal$. The linear inverse problem in Eq.(3.5) can be rewrite as QUBO Hamiltonian as

$$\mathcal{H}(\mathbf{x}) = -\sum_{(i,j)} J_{ij} x_i x_j - \sum_i h_i x_i, \tag{3.6}$$

where

$$J = -A^\intercal A - \lambda I = -\begin{pmatrix} 29+\lambda & 11 \\ 11 & 10+\lambda \end{pmatrix},$$

iteration	σ	Hq	H_pot
0	[[-1, 1], [1, -1], [1, -1]]	[183.441, -70.708, -41.892]	[169.033, -56.3, -56.3]
1	[[1, 1], [1, -1], [1, -1]]	[14.367, -70.708, -41.892]	[14.367, -56.3, -56.3]
2	[[1, -1], [1, -1], [1, -1]]	[-65.694, -70.708, -41.892]	[-56.3, -56.3, -56.3]
3	[[1, -1], [1, -1], [1, -1]]	[-65.694, -70.708, -41.892]	[-56.3, -56.3, -56.3]
4	[[1, -1], [1, -1], [1, -1]]	[-65.694, -70.708, -41.892]	[-56.3, -56.3, -56.3]

Table 1: The solution of each iteration using SQA algorithm to solve 2×2 linear systems in Example 1

and

$$\mathbf{h} = 2\mathbf{y}^\intercal A = \begin{pmatrix} 22 & 20 \end{pmatrix}.$$

Then, we can use SQA algorithm to solve this simple 2×2 linear system, and the solution of each iteration is shown in Table 1. The linear system shown in Eq.(3.5) is used a 3 Trotter slices Ising model and 4 iterations for SQA algorithm. Each Trotter slice have the same spin configuration which shown as the same values at step 2 of the iterations.

3.2 Floating-Point Calculation

In previous section, the linear inverse problem with binary variables can be easily to rewritten as a QUBO problem. However, in practice problem, the parameters always be floating point. In this section, we will express floating point operations in QUBO problem. In order to input our SQA algorithm more convinance, we will be express floating point operations in Ising model directly.

For floating point division to R bits of resolution, the graph G is in fact just the fully connected graph $K_R = (V_R, E_R)$. 4-bits of resolution is called a nibble and 8-bits is called a word. For example, a 2×2 linear system with floating variables will be used 4-bits of resolution to solve, which means each of variables would be used a 4 points of fully conncected graph to shown its value. The motivation of using R-bit binary points to approximate a floating point is used the infinite geometric series $\sum_{r=0}^{\infty} 2^{-r} = 2$. For any real value $\chi_i \in [0, 2)$, the binary representation accurate to R-bit binary format can be expressed by $[q_0.q_1q_2 \dots q_{R-1}]$, which can be shown as

$$\chi_i = \sum_{r=0}^{R-1} 2^{-r} q_r^i, \tag{3.7}$$

where $q_r^i \in \{0, 1\}$. Using $q_i = \frac{1}{2}\sigma_i + \frac{1}{2}$, Eq.(3.7) can be rewrite as

$$\chi_i = \sum_{r=0}^{R-1} 2^{-(r+1)} \sigma_r^i + \sum_{r=0}^{R-1} 2^{-(r+1)}, \tag{3.8}$$

and using $x_i = c\chi_i - d$, $\chi_i \in [0, 2)$ can be extend to $x_i \in [-d, 2c - d)$, and we have

$$\begin{aligned} x_i &= c \sum_{r=0}^{R-1} 2^{-(r+1)} \sigma_r^i + c \sum_{r=0}^{R-1} 2^{-(r+1)} - d \\ &= c \sum_{r=0}^{R-1} 2^{-(r+1)} \sigma_r^i + c_1, \end{aligned} \tag{3.9}$$

where $c_1 = c \sum_{r=0}^{R-1} 2^{-(r+1)} - d$.

The objective function of linear inverse problem can be shown as

$$f(\mathbf{x}) = \sum_{(i,j)} \hat{A}_{ij} x_i x_j - 2 \sum_{(i,j)} y_j A_{ji} x_i + \sum_i y_i^2, \tag{3.10}$$

where $x_i \in \mathbb{R}$. We try to recast Eq.(3.10) in the Hamiltonian form of Ising model

$$\mathcal{H}_{\text{Ising}} = -\sum_{(i,j)} a_r^i \sigma_r^i - \sum_{(i,j,r,s)} b_{rs}^{ij} \sigma_r^i \sigma_s^j. \tag{3.11}$$

For first part of Eq.(3.10), we have

$$\begin{aligned} H_1 &\equiv \sum_{(i,j)} \hat{A}_{ij} x_i x_j \\ &= \sum_{(i,j)} \hat{A}_{ij} \left(c \sum_{r=0}^{R-1} 2^{-(r+1)} \sigma_r^i + c_1 \right) \left(c \sum_{r'=0}^{R-1} 2^{-(r'+1)} \sigma_{r'}^j + c_1 \right) \\ &= c^2 \sum_{(i,j,r,r')} 2^{-(r+r'+2)} \hat{A}_{ij} \sigma_r^i \sigma_{r'}^j + 2cc_1 \sum_{(i,j,r)} 2^{-(r+1)} \hat{A}_{ij} \sigma_r^i + c_1^2 \sum_{(i,j)} \hat{A}_{ij}. \end{aligned}$$

For second part of Eq.(3.10), we have

$$\begin{aligned} H_2 &\equiv -2 \sum_{(i,j)} y_j A_{ji} x_i \\ &= -2 \sum_{(i,j)} y_j A_{ji} \left(c \sum_r 2^{-(r+1)} \sigma_r^i + c_1 \right) \\ &= -2c \sum_{(i,j,r)} 2^{-(r+1)} y_j A_{ji} \sigma_r^i - 2c_1 \sum_{(i,j)} y_j A_{ji}. \end{aligned}$$

Adding H_1 and H_2 gives

$$\begin{aligned} H &= H_1 + H_2 \\ &= \sum_{(i,j,r,r')} c^2 \cdot 2^{-(r+r'+2)} \hat{A}_{ij} \sigma_r^i \sigma_{r'}^j \\ &+ \sum_{(i,j,r)} (2cc_1 \cdot 2^{-(r+1)} \hat{A}_{ij} - 2c \cdot 2^{-(r+1)} y_j A_{ji}) \sigma_r^i + const. \\ &= \sum_{(i,j,r,r')} c^2 \cdot 2^{-(r+r'+2)} \hat{A}_{ij} \sigma_r^i \sigma_{r'}^j \\ &+ \sum_{(i,r)} \left(\sum_j 2cc_1 \cdot 2^{-(r+1)} \hat{A}_{ij} - 2c \cdot 2^{-(r+1)} y_j A_{ji} \right) \sigma_r^i + const. \end{aligned}$$

The Ising terms are therefore

$$a_r^i = -2cc_1 \cdot 2^{-(r+1)} \sum_j \hat{A}_{ij} + 2c \cdot 2^{-(r+1)} \sum_j y_j A_{ji} \tag{3.12}$$

$$b_{rr'}^{ij} = -c^2 \cdot 2^{-(r+r'+2)} \hat{A}_{ij}. \tag{3.13}$$

Here, the σ_r^i shows in the 2-dimensional indices, where $i = \{0, \ldots, N-1\}$ and $r = \{0, \ldots, R-1\}$. However, in last week SQA algorithm, we use the 1-dimensional index σ_i. Next, we use the row-major linear index mapping the indices between 2-dimensional and 1-dimensional. Let

$l(i,j) = i \cdot R + r$, then $i_l = \lfloor l/R \rfloor$, and $r_l = l \mod R$. The Ising terms are therefore

$$a_l = -2cc_1 \cdot 2^{-(r_l+1)} \sum_j \hat{A}_{i_l j} + 2c \cdot 2^{-(r_l+1)} y_j A_{j i_l} \tag{3.14}$$

$$b_{ll'} = c^2 \cdot 2^{-(r_l + r_{l'} + 2)} \hat{A}_{i_l i_{l'}}, \tag{3.15}$$

and using geometric series and Eq.(3.9), the solution of floating-point linear system can be solve as

$$x_{i_l} = c \sum_{r_l} 2^{-(r_l+1)} \sigma_l + c \sum_{r_l} 2^{-(r_l+1)} - d. \tag{3.16}$$

Hence, we have

$$\mathcal{H}_{\text{Ising}} = -\sum_{l=0}^{N\cdot R-1} a_l \sigma_l - \sum_{l=0}^{N\cdot R-1} \sum_{l'=0}^{N\cdot R-1} b_{ll'} \sigma_l \sigma_{l'}. \tag{3.17}$$

Example 2 *Given a simple linear system with floating-point variables,*

$$\begin{pmatrix} 0.5 & 1.5 \\ 1.5 & 0.5 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1.0 \\ 0.0 \end{pmatrix}, \tag{3.18}$$

where $\mathbf{x} = [-0.25, 0.75]^\intercal$.

In this example, 4-bits of resolution is used to solve which means 8 spin nodes used in Ising model. First, using Eq.(3.14) and (3.15), the Ising terms in Eq.(3.17) can be solve as

$$h(a_l) = \begin{bmatrix} 2.00 & 1.00 & 0.50 & 0.25 & 4.00 & 2.00 & 1.00 & 0.50 \end{bmatrix}^\intercal,$$

and

$$J(b_{ll'}) = \begin{bmatrix} -2.50 & -1.25 & -0.62 & -0.31 & -1.50 & -0.75 & -0.38 & -0.19 \\ -1.25 & -0.62 & -0.31 & -0.16 & -0.75 & -0.38 & -0.19 & -0.09 \\ -0.62 & -0.31 & -0.16 & -0.08 & -0.38 & -0.19 & -0.09 & -0.05 \\ -0.31 & -0.16 & -0.08 & -0.04 & -0.19 & -0.09 & -0.05 & -0.02 \\ -1.50 & -0.75 & -0.38 & -0.19 & -2.50 & -1.25 & -0.62 & -0.31 \\ -0.75 & -0.38 & -0.19 & -0.09 & -1.25 & -0.62 & -0.31 & -0.16 \\ -0.38 & -0.19 & -0.09 & -0.05 & -0.62 & -0.31 & -0.16 & -0.08 \\ -0.19 & -0.09 & -0.05 & -0.02 & -0.31 & -0.16 & -0.08 & -0.04 \end{bmatrix},$$

substitute J, and h into SQA algorithm, the results are solved as

$$\sigma = \begin{bmatrix} -1 & 1 & 1 & 1 & 1 & -1 & 1 & 1 \end{bmatrix}^\intercal,$$

and using geometric series and Eq.(3.9), the solution of linear system in Example 2 is

$$\mathbf{x} = [-0.25, 0.75]^\intercal.$$

Here, the Hamiltonian of transverse-field Ising model through the iteration are shown as Figure 7. Since has a higher temperature at the beginning of the iteration, the Hamiltonian of transverse-field Ising model can be increase, and then decrease to the lowest Hamiltonian at the end of the iteration.

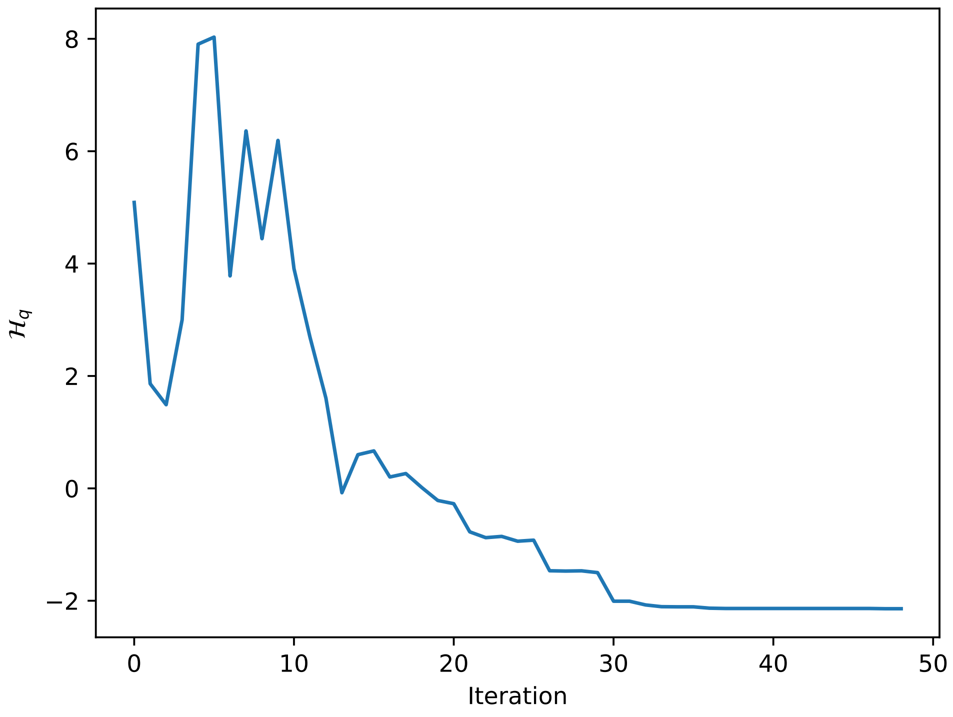


Figure 7: The Hamiltonian of the transverse-field Ising model for solving floating-point linear systems using the SQA algorithm through iteration.

4 Neural Network Structure Optimization

The QUBO problem is the optimal formulation for QA-based solving. However, not all problems can be reformulated as QUBO problems or the Ising model. This implies that not all problems can be solved in a quantum annealer. SQA simulates quantum properties on classical computers without being restricted from solving the QUBO problem or the Ising model. In this section, we employ the SQA algorithm to perform neural network pruning to optimize the structure of partially connected neural networks.

4.1 Introduction to Network Pruning

Deep learning in artificial intelligence has witnessed rapid development over the past decade, significantly influencing numerous industries. Deep learning algorithms can potentially make more informed decisions or accurate predictions [23][46]. The remarkable ability of deep learning is mainly due to the ability to build large neural networks. In order to obtain satisfactory performance, the neural network requires training, encompassing aspects such as initialization [17][20], backpropagation [21], and gradient updates [8][11][27]. Building a deeper neural network requires increasing the number of layers and nodes, leading to a gradual increase in the computational complexity of deep learning methods. Simultaneously, as the model size grows, the demand for storage space increases, increasing costs. To make neural networks work for devices with constrained computing power or storage capacity, the endeavour to compress the neural networks while preserving their good performance has attracted increasing attention.

Currently, there exist four primary methods for compressing neural networks: (1) Decreasing computational and storage demands by reducing the bit-width of reserved weight values [54]; (2) Reducing computational load through layer decomposition or simplification of activation functions within the neural network [55]; (3) Employing small neural networks for subtasks instead of large-scale ones [22]; (4) Reducing connections within neural networks—namely, through pruning—to transform fully connected structures into sparse neural networks [32][34][35]. In our thesis, we will specifically focus on the last approach. As an effective compression method, pruning removes redundancy in over-parameterized neural networks, achieving storage and computation savings without imposing significant performance penalties. The concept of neural network pruning dates back to 1989 when LeCun et al. [33] introduced the idea of removing insignificant weights from the network as a part of the pruning process. The neural network is composed of nodes and edges. Pruning mainly reduces the computational complexity by cutting off unimportant branches or nodes to reduce the number of neural network parameters. Pruning techniques can be further categorized into structured and unstructured pruning methods, guided by the structure characteristics of neural network [6]. Unstructured pruning involves zeroing out insignificant parameters based on heuristics such as weight values. Structured pruning involves removing substructures (e.g., layers) based on optimized importance scores or heuristics.

In much of the literature, less significant weights are commonly associated with lower weight values. For instance, Han et al. [19] proposed Threshold Pruning, where edges with an absolute weight value below a predefined threshold are pruned. This process is performed using a pre-trained network. After removing the smaller weights, the remaining weights are retrained using the dataset. This approach, referred to as the min-K method [32], involves pruning the weight parameters with the K smallest absolute values. However, following experiments in this report revealed the need to correct the notion that smaller weights are invariably less significant.

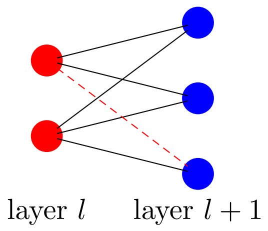
The remainder of this section is organized as follows. In Section 4.2, the details of using SQA algorithm to prune neural network will be introduced. In Section 4.3, some experiments is designed to evaluate the effectiveness of the SQA-based pruning method. MNIST and FASHION datasets will be used to train our neural network.

4.2 SQA-based Pruning Technology

Simulated quantum annealing, a heuristic approach, can address diverse optimization problems on classical computers by emulating the exploratory mechanisms employed by quantum annealers. This technique offers a pathway to solving intricate challenges, such as the neural network pruning problem. However, the practical application of the SQA algorithm requires some customization of its components.

4.2.1 Pruning Approach

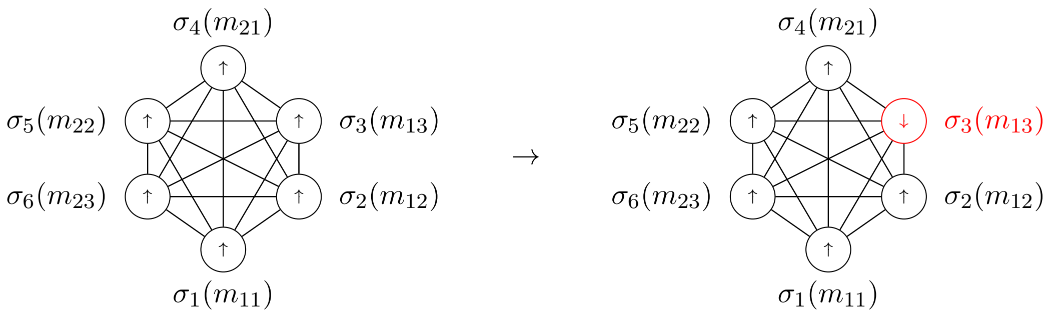
For the network structure optimization problem, we consider using a mask matrix M_l at l-th layer. For a fully connected neuron network, the mask matrix M_l is filled with ones in all positions for every layer. Suppose the edge between i-th node in one hidden layer and j-th node in the next hidden layer would be pruning. In that case, one will be replaced by zero at location (i, j) in the corresponding mask matrix M_l. In Figure 8, a two layers neural network structure is shown for pruning. If the edge between 1-st node in layer l to 3-rd node in layer $l + 1$ is pruned, then one would be replaced by zero at location $(1, 3)$ in the mask matrix.



(a) Network

$$\begin{array}{c} \text{Weight} \\ \begin{array}{|c|c|c|} \hline w_{11} & w_{12} & \color{red}{w_{13}} \\ \hline w_{21} & w_{22} & w_{23} \\ \hline \end{array} \end{array} \bullet \begin{array}{c} \text{Mask} \\ \begin{array}{|c|c|c|} \hline 1 & 1 & \color{red}{0} \\ \hline 1 & 1 & 1 \\ \hline \end{array} \end{array}$$

(b) Weight and Mask Matrix



(c) Spin Direction

Figure 8: Illustration for the implementation of pruning in two layers, the black lines show the links are connected between two nodes, and the red dashed line shows the link is disconnected between two nodes. The edge between 1-st node in layer l to 3-rd node in layer $l + 1$ is pruned, and one is replaced by zero at location $(1, 3)$ in the mask matrix.

In the SQA algorithm, the connection between two layers is shown as the direction of the spin. The network has two nodes in layer l and three nodes in layer $l + 1$, which means this network has six edges if the network is fully connected. For this network, a six-spin configuration can be constructed to correspond to each edge, which shown in Figure 8c. If the node spin upward, the corresponding edge would be connected. If the node spin downward, the corresponding edge would be disconnected. If one node spin swaps from upward to downward, the corresponding edge would be pruning. If one node spin swaps from downward to upward, the corresponding edge would be reconnected.

4.2.2 Neighborhood Rule

One of the most important components of SQA is defining the neighbourhood rule in the pruning network. The choice of the neighbourhood rule indicates the possible moves the random walk

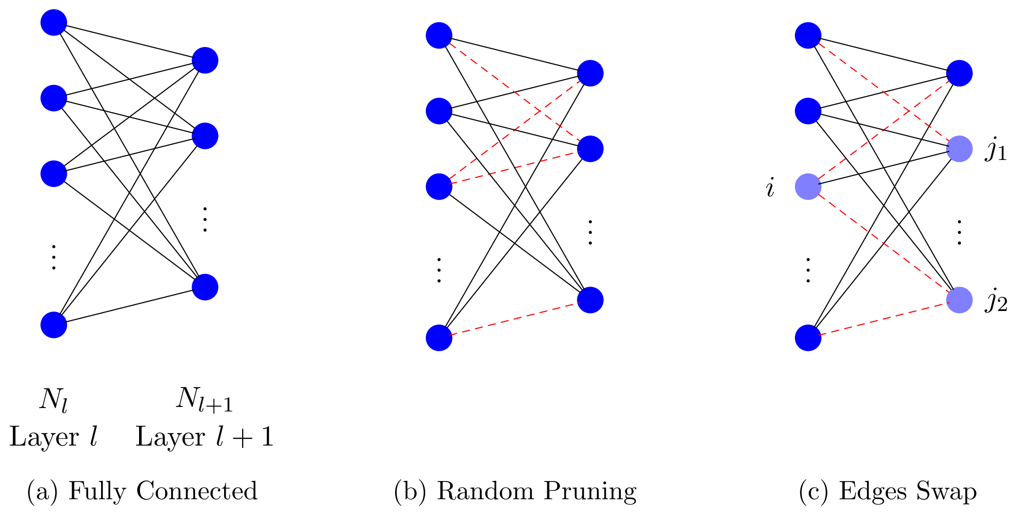


Figure 9: Illustration for our proposed methodology where the neighbourhood rule of SQA-based pruning method mainly works on the edge pair selection. For a fully connected neural network (a), randomly pruning a specific number of edges (b) and then using the neighbourhood rule to swap edges (c).

can make. It affects the convergence of the annealing process. In a sparse connected neural network, two edge sets can be defined, where one set contains the edges to which two nodes are connected, called the connected set, and the other set contains the edges that are disconnected, called the disconnected set. For the neighbourhood rule, one edge is chosen randomly from the connected set to disconnect it, and one edge is chosen randomly from the disconnected set to connect it. For a fully connected neural network (Figure 9a), randomly pruning a specific number of edges (Figure 9b) and then using the neighbourhood rule to swap edges (Figure 9c). In the SQA algorithm, one spin is swapped from upward to downward, and one is swapped from downward to upward. For example, two spin configurations are defined as $\sigma_{\text{new}} = [1, 1, -1, 1]^\intercal$ and $\sigma = [1, 1, 1, -1]^\intercal$, then σ_{new} is called one of the neighborhood of σ.

For a large and deep neural network, it is only switching one pair of edges of connected and disconnected in one pruning process that needs a large number of iteration steps, which is an expensive computational cost. A common approach is to choose more edges in connected and disconnected sets in one pruning process and switch them.

4.2.3 Objective Hamiltonian

Suppose that a fully connected neural network $\mathcal{N}(\cdot)$, a loss function $\mathcal{L}(\cdot)$, trianing the neural network is defined as

$$\{W_l^*, \mathbf{b}_l^*\}_{l=1}^k = \underset{\{W_l, \mathbf{b}_l\}_{l=1}^k}{\arg\min}\ \mathcal{L}\left(\mathcal{N}\left(\mathcal{T}_{\text{train}} | \{W_l, \mathbf{b}_l\}_{l=1}^k, \mathcal{T}_{\text{train}}^{\text{labels}}\right)\right), \tag{4.1}$$

where $W_l, \mathbf{b}_l$ are the parameters of neural network. The training process is to find the weights and biases to minimise the loss function of the neural network. In our neural network structure optimisation problem, mask matrics $\{M_l\}_{l=1}^k$ are introduced. The pruning process is to find the optimal mask matrics to minimise the loss function of the neural network, which has already been trained.

In Section 2.3.2, QA for solving the Ising model has been introduced. The objective Hamiltonian for QA can be defined as two parts, problem Hamiltonian and transverse-field Hamiltonian. The objective Hamiltonian for neural network structure optimisation problems can be defined as

$$\mathcal{H}(t) = \mathcal{H}_{\text{Loss}} + \Gamma(t)\mathcal{H}_{\text{TF}} \tag{4.2}$$

where $\mathcal{H}_{\text{Loss}} = \mathcal{L}\left(\mathcal{N}\left(\mathcal{T}_{\text{train}} | \{M_l\}_{l=1}^k, \mathcal{T}_{\text{train}}^{\text{labels}}\right)\right)$ is the loss function of neural network with respect of mask matrics, which defined as problem Hamiltonian. The corresponding Hamiltonian can be derived by the Suzuki-Trotter formula as

$$\mathcal{H}(t) = \sum_{k=1}^{M} \left(\frac{1}{M} \mathcal{H}_{\text{Loss}} - \gamma(t) \sum_i \sigma_i^k \sigma_i^{k+1} \right), \tag{4.3}$$

where M is the total number of Trotter slices (or called replicas), σ_i^k is used to denote a classical Ising spin at the ith site on the kth Trotter slice, and the time-dependent coefficient of the term $\gamma(t)$ is given by

$$\gamma(t) = \frac{T}{2} \ln \left(\coth \left(\frac{\mathit{\Gamma}(t)}{MT} \right) \right). \tag{4.4}$$

Here $\gamma(t)$ represents the strength of couplings between Trotter slices.

4.2.4 Accepted Probability

As mentioned in Section 2.4, the decision to accept or reject a move or a step of the random walk in the SQA algorithm uses Metropolis-Hastings (MH) algorithm. The most critical step in the MH algorithm is to calculate the accepted probability according to the probability of Boltzmann distribution. At each Trotter slice, the accepted probability of a move or a step of the random walk is calculated as follows:

$$\mathit{AcceptProb}(T, \Delta) = \min \left(1, e^{-\Delta \cdot M/T} \right), \tag{4.5}$$

where T is the temperature at one annealing process, M is the number of Trotter slices, and $\Delta = \mathcal{H}_{\text{indi}}(\sigma') - \mathcal{H}_{\text{indi}}(\sigma)$ is the increase of the 'energy' for individual Trotter slice which is calculated as

$$\mathcal{H}_{\text{indi}}(\sigma) = \mathcal{H}_{\text{Loss}}(\sigma) - \gamma \sum_i \sigma_i^k \sigma_i^{k+1}. \tag{4.6}$$

At each step, a uniform random number p is generated between $[0, 1)$ and compared with $\mathit{AcceptProb}(T, \Delta)$. If $p < \mathit{AcceptProb}(T, \Delta)$, accept this move; otherwise, rejected.

4.2.5 Converge Condition

The main factors that influence the performance of SQA in network pruning problems are four parameters: temperature, transverse field, temperature decrease rate, and the number of Trotter slices.

Temperature Instead of using a fixed value of the initial temperature T_0 in our experiments, an approach has been introduced in Section 2.2.2 above to initialize the temperature T_0. The details of the approach to initialize T_0: Suppose that we want to prune $r\%$ edges in a given neural network; an initial mask matrix is randomly generated and calculated the loss function for this sparse network $\mathcal{L}(M)$, and then randomly choose 1000 neighbour of the initial mask matrix using the neighbourhood rule and calculate their loss function $\mathcal{L}(M')$. The initial temperature is calculated by $T_0 = \max(\mathcal{L}(M')) - \min(\mathcal{L}(M'))$. As we mentioned in Section 2.2.2, a common approach for the cooling schedule is using geometric cooling, $T(t) = \eta T(t-1)$, where η is the cooling rate to control the decrease of the temperature. In our experiments, the geometric cooling schedule is used to control the temperature decrease.

Transverse Field In Section 2.4, the converge condition for the transverse field of the final Ising Hamiltonian has been introduced. The transverse field $\mathit{\Gamma}(t)$ is calculated at a fixed temperature T, starts at a high value, and decreases to zero according to time t. In our experiments, temperature $T(t)$ is calculated as the geometric cooling schedule for each iteration of the annealing process. In order to consider the relationship between the transverse field and temperature with iteration (or time). Transverse field $\Gamma(t)$ is considered to calculated as follows:

$$\Gamma(t) = MT(t)\tanh^{-1}\left(\frac{1}{(4N)^{1/2N}}\right). \tag{4.7}$$

Trotter Slice As we mentioned in Section 2.4, by taking a large enough number of Trotter slice M and a small enough temperature T, we have a high probability of finding the ground state of Hamiltonian. However, too large the number of Trotter slices M obviously would have a large computational cost. The following section will discuss the performance of SQA based method under different numbers of M.

Here, using the geometric cooling schedule to control the temperature and Eq.(4.7) to control the transverse field Γ, the time-dependent coefficient of the term $\gamma(t)$ which is shown in Eq.(4.4) are shown in Figure 10. The cooling rate η is used to control the decrease of the temperature T, and according to Eq.(4.7), η also control the decrease of the transverse field Γ. If the choice of η is too small, which means the temperature and transverse field decrease too quickly, the random walk of the Markov chain cannot implement the 'tunnelling effect' of quantum annealing, and the convergence of the SQA algorithm cannot be guaranteed.

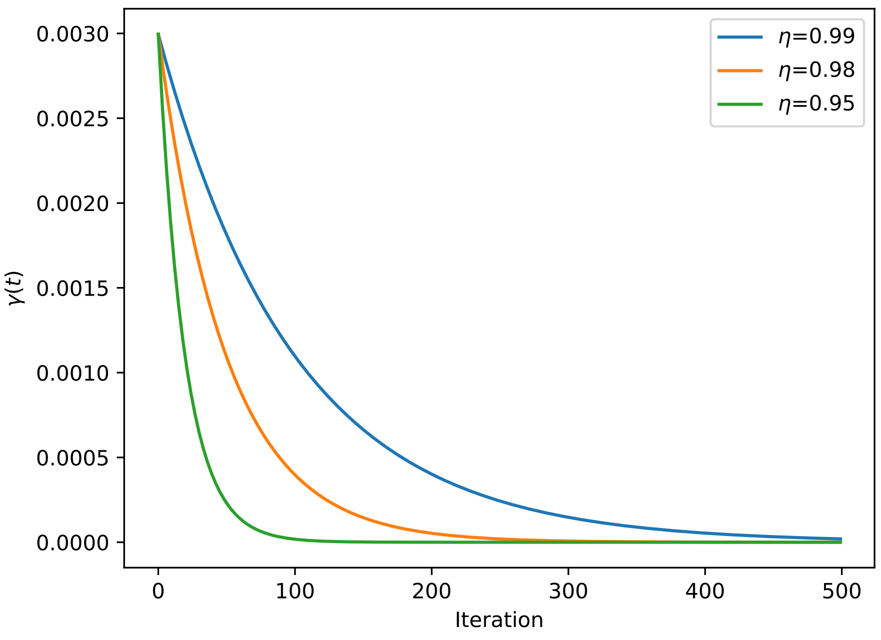


Figure 10: The time-dependent coefficient of the term $\gamma(t)$ is controlled by cooling rate η, the transverse field Γ is controlled by Eq.(4.7), and the temperature T is controlled by the geometric cooling schedule.

4.3 Experiments

According to the pruning strategy introduced above, which is to optimize Eq.(4.3) by applying SQA algorithm. The neural network in Figure 11, which has three hidden layers, is designed to evaluate the effectiveness of the SQA algorithm in network pruning problems. All the edges between the input, hidden, and output layers will be pruned. In order to evaluate the performance of the pruned network, two image datasets will be used in the experiments, namely MNIST and FASHION. MNIST is a classic image dataset of handwritten numbers from zero to nine. It is the most famous dataset used in Machine Learning. FASHION (or Fashion-MNIST) is a classic dataset image of ten different categories of products. FASHION is more complex than MNIST. The whole optimization process starts from a well-trained, fully connected neural network, which is done by gradient descent repeatedly.

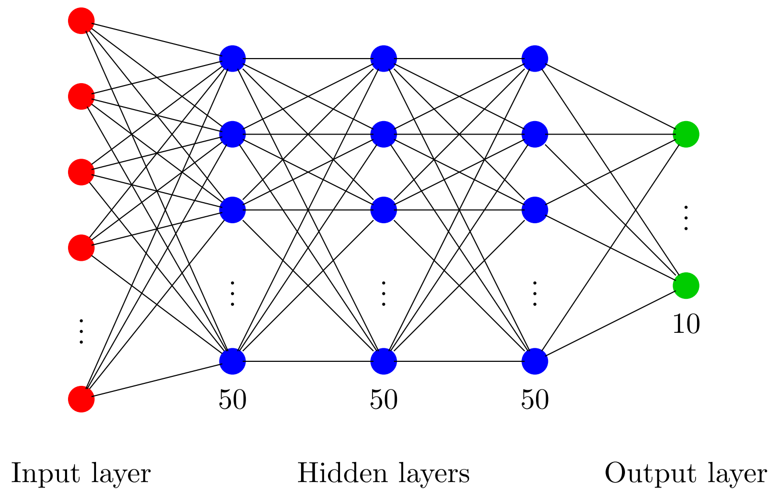
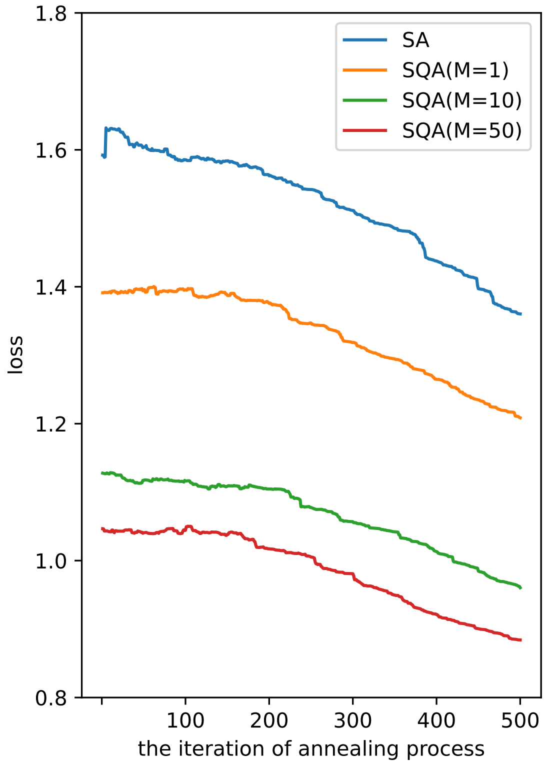


Figure 11: The neural network structure with three hidden layers. The purpose of this network is to evaluate the effectiveness of SA and SQA in network pruning. All the edges in the input, hidden, and output layers will be pruned.

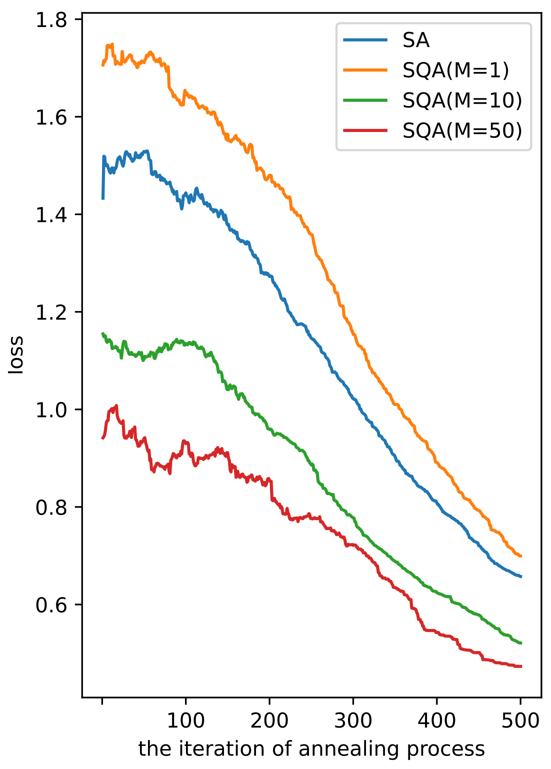
In this section, two parts of experiments will be focused on: the first part is to investigate the convergence of SQA-based method for solving neural network structure optimization problem, which will be shown in Section 4.3.1; the second part is to investigate the performance of SQA based method under different pruning scales for neural networks. Meanwhile, investigate the performance of SQA based pruning method, two results will be shown: the first is using SQA based method to pruning neural network without retraining this network to show the performance of our method in pruning itself; the second is to retrain this network after pruning. In order to realize the training of the sparse neural network, Kuo et al. [32] modified the full link function (nn.Linear) in Pytorch. They customized a class function (CustomizedLinear). The purpose is to involve the mask matrix, which, as mentioned above, is in the forward and backward propagation process and realize the training of a sparse neural network. Their retrained sparse nerual network method will be used in our experiments. All code used for these experiments is available on GitHub at https://github.com/itsWei/MScThesisPruning.

4.3.1 Convergence of SQA Method

Here, to evaluate the convergence of the SQA in the pruning process, the value of the loss function through the annealing process is shown in Figure 12a. The fully connected neural network is used in Figure 11, which is trained by the MNIST dataset. Randomly pruning 50% edges between all of the layers to be the initial state, and using SQA algorithm with three different numbers of Trotter Slices $M \in [1, 10, 50]$ to solve this network pruning problem. At each step of the iteration, the output, the value of the loss function, is not used the whole Hamiltonian, which is shown in Eq.(4.3), rather than using the minima of the loss function between each Trotter slices. In our experiments, at the end of the pruning process, which is not the same



(a) Switch one pair edges in one iteration



(b) Switch ten pair edges in one iteration

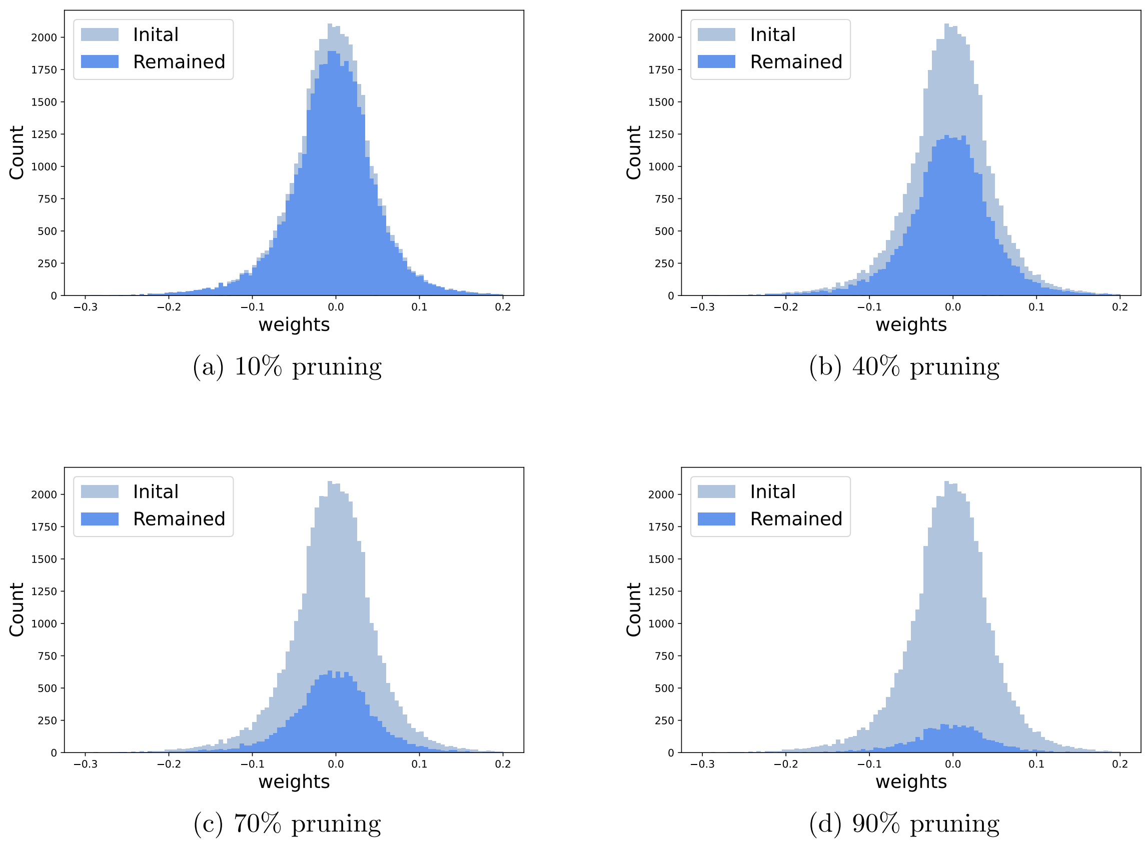
Figure 12: The loss function through the iteration using SQA with three different M and SA. Figure (a) shows the SQA algorithm using one pair of edges switching in one pruning process for the neighbourhood rule, and Figure (b) uses ten pairs of edges.

as other references, output the global minima of spin state for the whole process, the output using the minima spin state between each Trotter slices at the end of the pruning process. In Figure 12a, three of four different lines show using SQA algorithm with three different numbers of M, and the last line shows using standard SA algorithm-based method to be compared. From the three lines of SQA based method to pruning the network, a higher number of Trotter slices used in the algorithm, the performance shows be better. However, these three lines show the widely dispersed with large gaps, which cannot show any convergence. As we mentioned in Neighborhood Rule in Section 4, for a large neural network, only switching one pair of edges of connected and disconnected in one annealing process needs a large number of iteration steps, which expensive computational cost. Here, to show the convergence of the annealing process, the approach we use is to choose more edges in the connected set and disconnected set and to switch them.

In Figure 12b, switch ten pair edges of connected and disconnected in one annealing process, and same with Figure 12a, three of four different lines show using SQA algorithm with three different numbers of M. The last line shows using standard SA algorithm-based method to be compared. From the three lines of SQA based method to pruning the network, a higher number of Trotter slices used in the algorithm, the performance shows be better at the beginning of the annealing process. These lines converge to the same point at the end of the iteration. The pruning problem-solving using the SQA algorithm with one Trotter slice performs almost the same as solving using the standard SA algorithm mentioned above.

4.3.2 Visualization of the Weight Parameters after Pruning

In order to observe the performance of a neural network at different pruning percentages, the histogram plots shown in Figure 13 compare the value of weight parameters between fully connected neural networks with pruned, sparse-connected neural networks. The four plots in Figure 13 show the pruning process using the SQA-based method. In Figure 13a and 13b, which shows pruning 10% and 40% edges for the fully connected neural network, SQA-based pruning process trend to pruning the value of weights around 0. This trend is similar to the threshold pruning (min-K) method, in which the pruning criteria is to select those weights with an absolute magnitude smaller than a threshold. The histogram plots shown in Figure 20 compare the value of weight parameters between fully connected neural networks with pruned, sparse-connected neural networks which using min-K method to pruned. In a neural network, the weight of one edge between two nodes is close to 0, meaning no information is transferred from the node in the previous layer to the node in the next layer. The threshold pruning (min-K) method used to prune is to remove the edges whose weight value is smaller than a bound. However, the difference between the threshold pruning (min-K) method and our SQA-based method is that not all pruned edges value close to 0. In Figure 13c and 13d, the edges which weight value close to 0 also remained, which shows that in the SQA-based pruning method, these edges are also useful for decreasing the loss value in the annealing process.



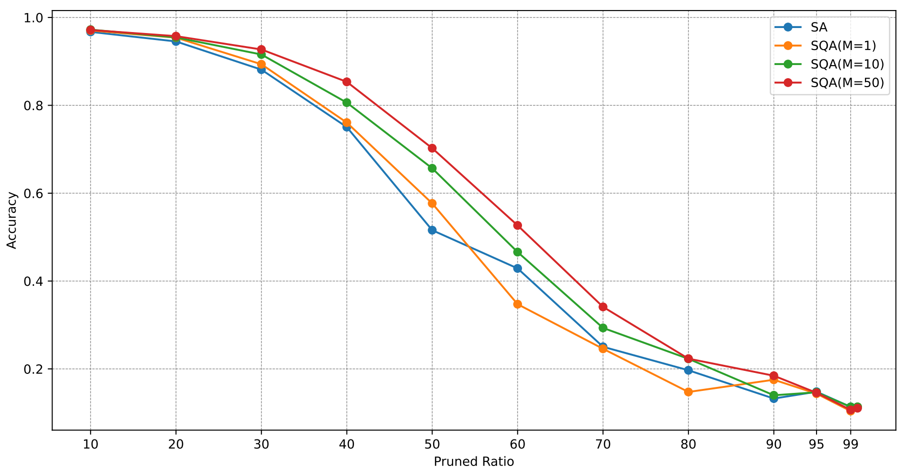
(a) 10% pruning (b) 40% pruning

(c) 70% pruning (d) 90% pruning

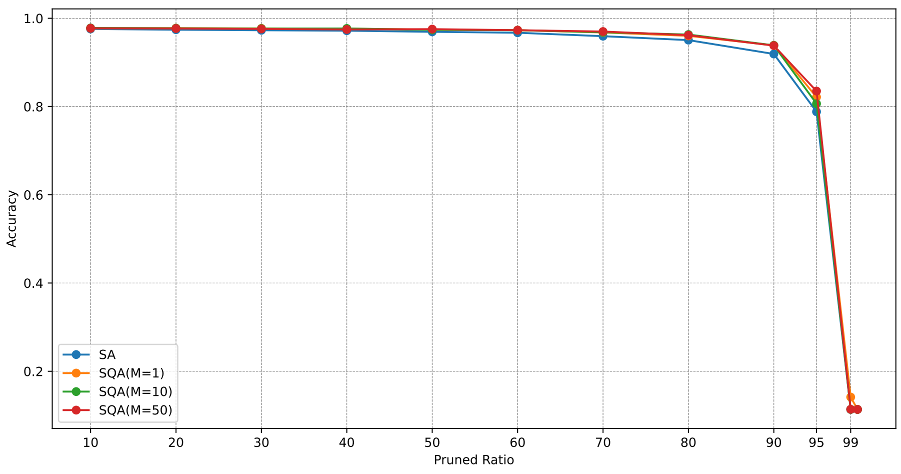
Figure 13: The histogram of the total weight parameters in the different pruning ratios using the SQA-based pruning method, the light blue represents the initial weight parameters without pruning, and the other represents the weight parameters after pruning, where 10% pruning in (a), 40% pruning in (b), 70% pruning in (c), and 90% pruning in (d).

4.3.3 Performance Trend under Different Pruning Scales

In this section, the performance from standard simulated annealing and three different number of Trotter slices simulated quantum annealing based method at various pruning scales shown



(a) The pruned network without using retrained.

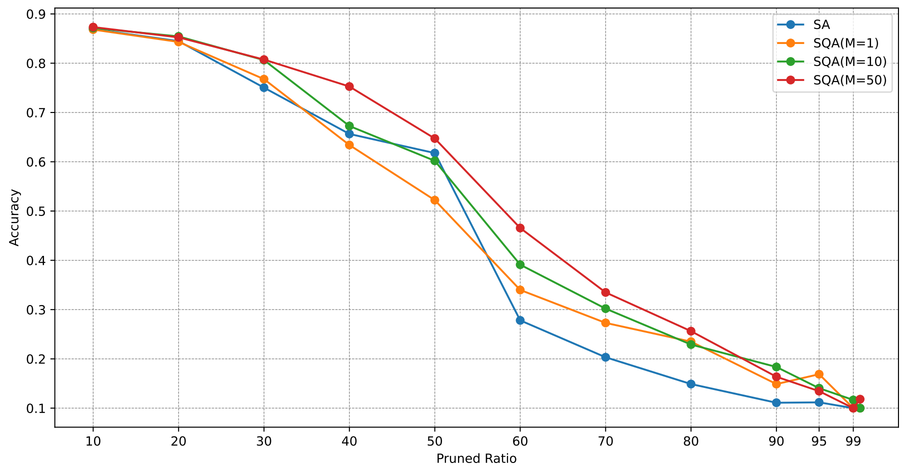


(b) The pruned network using retrained.

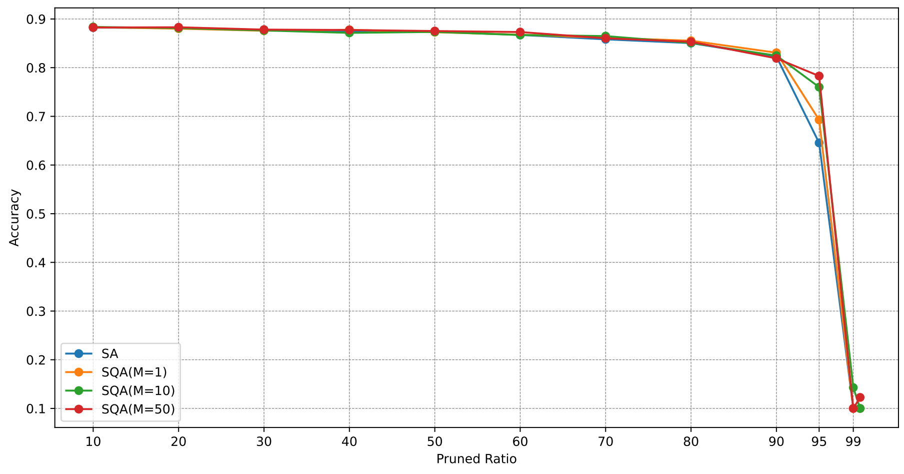
Figure 14: The accuracy corresponds to the different pruning ratios of the fully connected neural network trained using the MNIST dataset. The pruned network without using retrained in (a), and using retrained in (b).

in Figures 14b. The MNIST dataset will be used to train the fully connected neural network shown in Figure 11. First, the accuracy of the neural network after different percentages of pruning using difference methods shown in Figure 14a. When the number of Trotter slices M increases, the optimal configurations with lower loss values can be better to find. When the number of Trotter slices is small, the line of loss and accuracy at different pruning ration has more fluctuating, which also shows that SQA can find the ground state with high probability with large enough M as we mentioned in Section 2.4.

The SQA-based method for pruning shows its potential to find the optimal configuration when all weights and biases parameters are fixed. After the SQA-based pruning process, the sparse neural network is retrained using MNIST dataset. Moreover, the accuracy of the neural network after different percentages of pruning using different methods is shown in Figure 14b. According to the comparison of accuracy values at different percentages of pruning in Figure 14b with the accuracy, which directly using SQA-based pruning method without retrained, in Figure 14a, most of the lost accuracy can be recovered. Even though the fully connected neural network pruning almost 90 % of edges, the accuracy can be recovered to closed accuracy with non-pruning. At the end of the pruning ratio like 95% in Figure 14b, the accuracy of the sparse



(a) The pruned network without using retrained.



(b) The pruned network using retrained.

Figure 15: The accuracy corresponds to the different pruning ratios of the fully connected neural network trained using the FASHION dataset. The pruned network without using retrained in (a), and using retrained in (b).

connected neural network can be recovered to 80 % accuracy.

However MNIST is too simple, and the accuracy rate of many deep learning algorithms on the test set has reached 99.6%. Here, a more complex dataset will be used which is FASHION. FASHION (or Fashion-MNIST) is an image dataset that replaces the MNIST handwritten digit set. The fully connected neural network shown in Figure 11 will be trained using FASHION dataset. The accuracy of the neural network after different percentages of pruning using difference methods shown in Figure 15a and the accuracy of the sparse neural network which retrained using FASHION dataset shown in Figure 15b. According to the comparison of accuracy values at different percentages of pruning in Figure 15b with the accuracy, which directly using SQA-based pruning method without retrained, in Figure 15a, most of the lost accuracy can be recovered.

Both MNIST and FASHION are associated with ten different classes of objects. If the accuracy of the network is close to 10%, it is equivalent to random guessing results. In Figure 14b and 15b, after pruning 95% of the network and after retraining, the accuracy can be recovered to about 80%. Meanwhile, it can be found that the larger M (Trotter Slices) is, the higher the accuracy after retraining is.

4.3.4 SQA Efficiency with Min-K Initialization

The neural network used in our experiments, which is shown in Figure 11, is three hidden layers and 50 neurons in each hidden layer. The min-K method performs well for pruning the edges with weights close to zero. However, for edges whose weights are not close to zero, whether an edge with a higher weight value is more critical than an edge with a lower weight value needs further investigation. SQA-based pruning method could help us to investigate the efficiency of pruning neural networks using the min-K method for the edges whose weights are not close to zero.

The histogram plots in Figure 21 show the value range of weight parameters between each layer of the fully connected neural network used in our experiments and trained by the MNIST dataset. In Figure 21a, most edges between the input layer and the hidden layer one have an absolute weight value close to zero, which also shows that most of the edges in this network are useless and could be pruned directly. The edges between the input layer and the hidden layer one have 784×50, and the value of the large edges close to zero would affect the performance of the SQA-based pruning method. In order to eliminate this effect, a smaller neural network will be used in our following experiments. The neural network has one hidden layer, which is shown in Figure 19.

In the previous section, the SQA-based method was used randomly for initialization. For an SQA-based pruning method with M Trotter slices, and to pruning $r\%$ edges for a neural network, initialization of each Trotter slice was used randomly to choose $r\%$ edges for the fully connected neural network. In this section, the initialization of the SQA-based method will be used min-K method. For a $r\%$ pruning problem, the $(r+p)\%$ edges will be chosen using the min-K method, and initialization of each Trotter slice will be used random to choose $r\%$ edges in its $(r+p)\%$ edges. The rest of the $p\%$ edges is used for each Trotter slice with different spin configurations. Here $p\%$ is chosen as 1%.

The accuracy of the neural network after different percentages of pruning using the SQA-based method with min-K initialization is shown in Figures 16 and 17, which uses the MNIST dataset to train the network in Figure 16, and using FASHION dataset to train the network in Figure 17. The two sparse connected neural networks are used the SQA-based method only without using two datasets to retrain. We can find that in Figure 16 when the pruning ratio is less than 60%, the sparse connected neural network trained by MNIST dataset, using SQA-based pruning method has the same accuracy with using min-K method. However, when the pruning

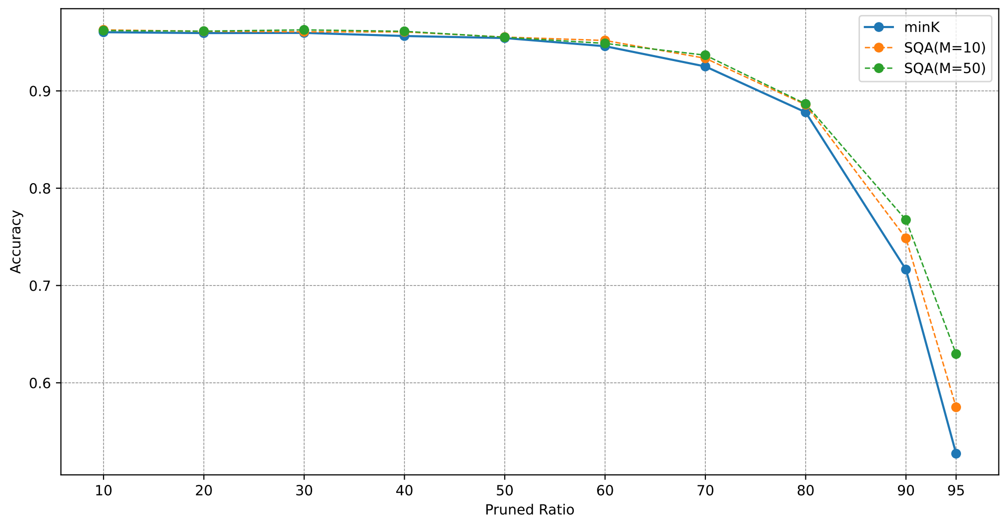


Figure 16: The SQA-based pruning method uses the min-K method to initialize, which implement in a small neural network with one hidden layer and trained by MNIST dataset.

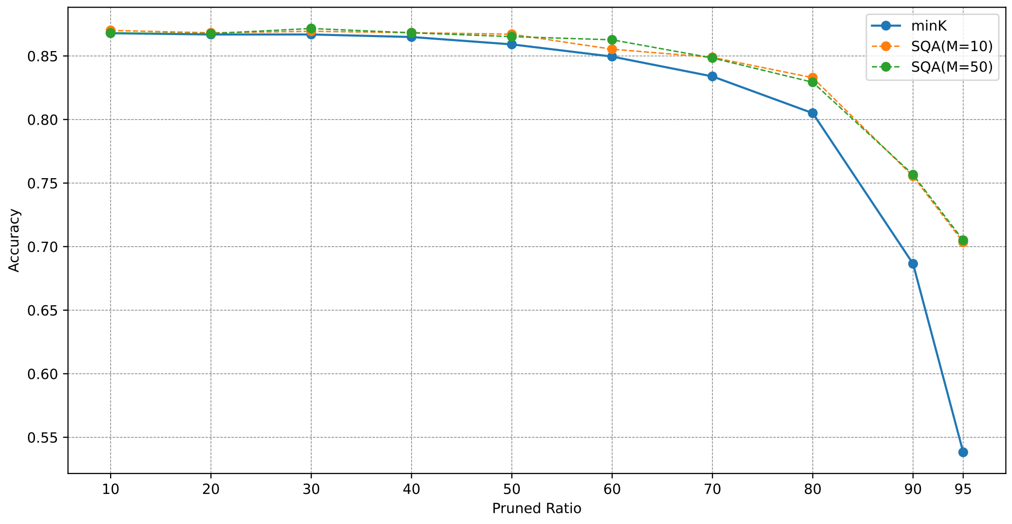
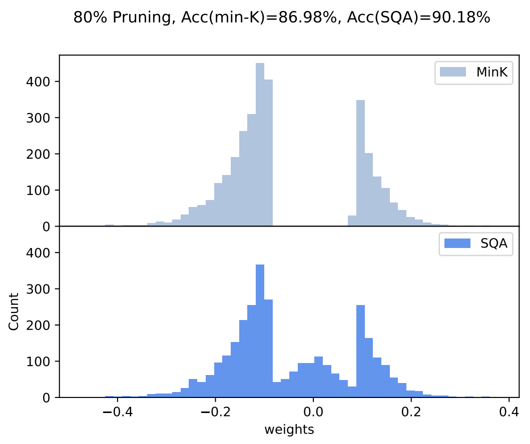


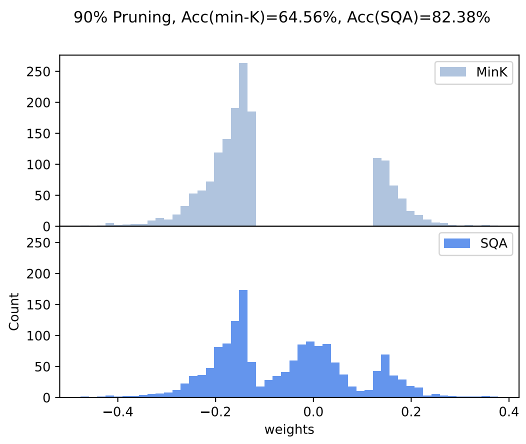
Figure 17: The SQA-based pruning method uses the min-K method to initialize, which implement in a small neural network with one hidden layer and trained by FASHION dataset.

ratio is more than 60%, the SQA-based method can find a better pruning state than the min-K method, which is shown as having higher accuracy. In Figure 17, the FASHION dataset trains the neural network, which is more complex than MNIST. When the pruning ratio is more than 50%, the SQA-based method can find a better pruning state than the min-K method, which shows higher accuracy. These experiments show that an edge with a higher weight value does not necessarily mean it is more critical than an edge with a lower weight value.

Meanwhile, the histogram of the total weight parameters in the different pruning ratios using the SQA-based and min-K methods is shown in Figure 18. In Figure 18a, the 80% edges in the network have been pruned. Use the min-K method for pruning, and 80% of the edges with smaller weights are pruned. The sparse connected neural network has 86.98% accuracy in the MNIST dataset. However, the SQA-based pruning method could find a higher accuracy connection for the same ratio of using the min-K method to prune the neural network. Figure 18 can also explain that an edge with a higher weight value does not necessarily mean it is more critical than an edge with a lower weight value.



(a) 80% pruning



(b) 90% pruning

Figure 18: The histogram of the total weight parameters in the different pruning ratios using the SQA-based pruning method and min-K method, the light blue represents the weight parameters using the min-K pruning method, and the other represents the weight parameters using the SQA-based pruning method with the min-K method initialization, where 80% pruning in (a), and 90% pruning in (b).

5 Conclusions

Within this thesis, we have implemented the Simulated Quantum Annealing (SQA) algorithm to tackle problems in scientific computing and machine learning. In Section 3, we initially explored rewriting and solving linear systems of equations into quadratic unconstrained binary optimization (QUBO) problems, which emerge as the most suitable formulations for the quantum annealer. We introduced the most straightforward linear system—employing binary variables—to be rewritten as a QUBO problem. Subsequently, we introduced floating-point calculations. Throughout this experiment, we demonstrate that when solving problems requiring greater precision or dealing with larger scales, more variables within the SQA algorithm would be used. This observation suggests that solving sizable optimization challenges on a real quantum annealer would demand an increased deployment of qubits. The number of qubits is also a significant recent limitation of real quantum computers.

Next, in Section 4, we investigate the application of SQA for neural network pruning. As we mentioned, rewriting the problem into a QUBO problem or Ising model is a prerequisite for solving it on a real quantum annealer. Hence, the challenge we initially confronted was encoding the objective function of the neural network pruning problem into a QUBO problem. We discovered that SQA, presented as a classical algorithm, solve the optimization problem without such restriction. The objective function of SQA can be expressed using various formulations. In Section 4, we demonstrate the effectiveness of the SQA algorithm in solving a problem that cannot be rewritten as a QUBO problem. For solving the neural network pruning problem, Kuo et al. [32] used a small neural network, and only the edges in hidden layers would use the SA algorithm to prune. In our work, we extend to the whole layers. All the edges in the input, hidden, and output layers were pruned. Our experiments found certain limitations in the standard SQA method for solving specific optimization problems. When a considerable number of weight parameters neared zero in the trained neural network, the effectiveness of the SQA-based pruning method was significantly worse than the min-K method. The rationale behind this phenomenon is apparent and was also discussed in the previous section. Consequently, we proposed the utilization of the min-K method as an initialization technique for the SQA-based pruning method, leading to improved performance compared to employing a random initialization. The SQA-based pruning method exhibited the ability to identify connections with higher accuracy when applying the min-K method for neural network pruning at the same pruning ratio. Furthermore, our experiments demonstrate that an edge with a lower weight value does not necessarily indicate greater significance than an edge with a higher weight value.

Our findings show that increasing the quantum behaviour of the algorithm yields better results, thus foreshadowing the possibility of achieving enhanced performance when the algorithm is eventually deployed on an actual quantum computing platform. Therefore, our future work is obvious.

Future Work

Within Section 4.2.3, the objective Hamiltonian was formulated based on the loss function of the neural network. However, transferring the loss function into a QUBO problem is complex, making the pruning problem infeasible for a solution on a quantum annealer. Our future work aims to devise an approach for mapping the neural network's loss function onto a QUBO problem. This would enable us to resolve our pruning problem or neural network problem by utilising an actual quantum computing platform.

References

[1] D. Aharonov, W. Van Dam, J. Kempe, Z. Landau, S. Lloyd, and O. Regev. Adiabatic quantum computation is equivalent to standard quantum computation. *SIAM review*, 50(4):755–787, 2008.

[2] C. Audet and J. E. Dennis. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2002.

[3] D. A. Battaglia, G. E. Santoro, and E. Tosatti. Optimization by quantum annealing: Lessons from hard satisfiability problems. *Phys. Rev. E*, 71:066707, Jun 2005.

[4] D. P. Bertsekas. *Constrained optimization and Lagrange multiplier methods.* Academic press, 2014.

[5] M. Born and V. Fock. Beweis des adiabatensatzes. *Zeitschrift für Physik*, 51(3-4):165–180, 1928.

[6] T. Chen, H. Zhang, Z. Zhang, S. Chang, S. Liu, P.-Y. Chen, and Z. Wang. Linearity grafting: Relaxed neuron pruning helps certifiable robustness, 2022.

[7] E. Crosson and A. W. Harrow. Simulated quantum annealing can be exponentially faster than classical simulated annealing. In *2016 IEEE 57th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 714–723, 2016.

[8] Y. Dauphin, H. De Vries, and Y. Bengio. Equilibrated adaptive learning rates for non-convex optimization. *Advances in Neural Information Processing Systems*, 28, 2015.

[9] D. Delahaye, S. Chaimatanan, and M. Mongeau. Simulated annealing: From basics to applications. *Handbook of metaheuristics*, pages 1–35, 2019.

[10] J. E. Dennis Jr and R. B. Schnabel. *Numerical methods for unconstrained optimization and nonlinear equations.* SIAM, 1996.

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(7), 2011.

[12] B. Dury and O. Di Matteo. A qubo formulation for qubit allocation. *arXiv preprint arXiv:2009.00140*, 2020.

[13] E. Farhi, J. Goldstone, S. Gutmann, J. Lapan, A. Lundgren, and D. Preda. A quantum adiabatic evolution algorithm applied to random instances of an np-complete problem. *Science*, 292(5516):472–475, 2001.

[14] E. Farhi, J. Goldstone, S. Gutmann, and M. Sipser. Quantum computation by adiabatic evolution. *arXiv preprint quant-ph/0001106*, 2000.

[15] J. Fliege, L. M. G. n. Drummond, and B. F. Svaiter. Newton's method for multiobjective optimization. *SIAM Journal on Optimization*, 20(2):602–626, 2009.

[16] Z. W. Geem, J. H. Kim, and G. V. Loganathan. A new heuristic optimization algorithm: harmony search. *simulation*, 76(2):60–68, 2001.

[17] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[18] F. Glover, G. Kochenberger, and Y. Du. A tutorial on formulating and using qubo models, 2019.

[19] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2016.

[20] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[21] Hecht-Nielsen. Theory of the backpropagation neural network. In *International 1989 Joint Conference on Neural Networks*, pages 593–605 vol.1, 1989.

[22] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015.

[23] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[24] K. Ikeda, Y. Nakamura, and T. S. Humble. Application of quantum annealing to nurse scheduling problem. *Scientific reports*, 9(1):12837, 2019.

[25] T. Kadowaki and H. Nishimori. Quantum annealing in the transverse ising model. *Phys. Rev. E*, 58:5355–5363, Nov 1998.

[26] Y. Kimura and H. Nishimori. Convergence condition of simulated quantum annealing for closed and open systems. *Physical Review A*, 106(6):062614, 2022.

[27] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[28] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[29] E. Knill. Quantum computing. *Nature*, 463(7280):441–443, 2010.

[30] B. H. Korte, J. Vygen, B. Korte, and J. Vygen. *Combinatorial optimization*, volume 1. Springer, 2011.

[31] S. Kumar, P. Jangir, G. G. Tejani, M. Premkumar, and H. H. Alhelou. Mopgo: A new physics-based multi-objective plasma generation optimizer for solving structural optimization problems. *IEEE Access*, 9:84982–85016, 2021.

[32] C. L. Kuo, E. E. Kuruoglu, and W. K. V. Chan. Neural network structure optimization by simulated annealing. *Entropy*, 24(3), 2022.

[33] Y. LeCun, J. Denker, and S. Solla. Optimal brain damage. In D. Touretzky, editor, *Advances in Neural Information Processing Systems*, volume 2. Morgan-Kaufmann, 1989.

[34] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, and L. Shao. Hrank: Filter pruning using high-rank feature map, 2020.

[35] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K.-T. Cheng, and J. Sun. Metapruning: Meta learning for automatic neural network channel pruning, 2019.

[36] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

[37] C. C. McGeoch. Adiabatic quantum computation and quantum annealing theory and practice.

[38] N. Mohseni, P. L. McMahon, and T. Byrnes. Ising machines as hardware solvers of combinatorial optimization problems. *Nature Reviews Physics*, 4(6):363–379, 2022.

[39] T. Okuyama, M. Hayashi, and M. Yamaoka. An ising computer based on simulated quantum annealing by path integral monte carlo method. In *2017 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–6, 2017.

[40] J. Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[41] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.

[42] B. W. Reichardt. The quantum adiabatic optimization algorithm and local minima. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 502–510, 2004.

[43] M. L. Rogers and R. L. Singleton. Floating-point calculations on a quantum annealer: Division and matrix inversion. *Frontiers in Physics*, 8, 2020.

[44] M. L. Rogers and R. L. Singleton Jr. Floating-point calculations on a quantum annealer: Division and matrix inversion. *Frontiers in Physics*, 8:265, 2020.

[45] H. Sakaguchi, K. Ogata, T. Isomura, S. Utsunomiya, Y. Yamamoto, and K. Aihara. Boltzmann sampling by degenerate optical parametric oscillator network for structure-based virtual screening. *Entropy*, 18(10), 2016.

[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[47] O. Schutze, E.-g. Talbi, G. T. Pulido, C. C. Coello, and L. V. Santana-Quintero. A memetic pso algorithm for scalar optimization problems. In *2007 IEEE Swarm Intelligence Symposium*, pages 128–134. IEEE, 2007.

[48] P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[49] S. Speziali, F. Bianchi, A. Marini, L. Menculini, M. Proietti, L. F. Termite, A. Garinei, M. Marconi, and A. Delogu. Solving sensor placement problems in real water distribution networks using adiabatic quantum computation. In *2021 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 463–464. IEEE, 2021.

[50] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11:341–359, 1997.

[51] T. D. Tambunan, A. B. Suksmono, I. J. Edward, and R. Mulyawan. Quantum annealing for vehicle routing problem with weighted segment. *arXiv preprint arXiv:2203.13469*, 2022.

[52] W. Van Dam, M. Mosca, and U. Vazirani. How powerful is adiabatic quantum computation? In *Proceedings 42nd IEEE symposium on foundations of computer science*, pages 279–287. IEEE, 2001.

[53] D. Volpe, G. A. Cirillo, M. Zamboni, and G. Turvani. Integration of simulated quantum annealing in parallel tempering and population annealing for heterogeneous-profile qubo exploration. *IEEE Access*, 11:30390–30441, 2023.

[54] K. Wang, Z. Liu, Y. Lin, J. Lin, and S. Han. Haq: Hardware-aware automated quantization with mixed precision. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8604–8612, 2019.

[55] X. Zhang, J. Zou, X. Ming, K. He, and J. Sun. Efficient and accurate approximations of nonlinear convolutional networks, 2014.

[56] H. Zheng, Y. Feng, and J. Tan. A hybrid energy-aware resource allocation approach in cloud manufacturing environment. *IEEE Access*, 5:12648–12656, 2017.

[57] R. Ziadi, R. Ellaia, and A. Bencherif-Madani. Global optimization through a stochastic perturbation of the polak–ribière conjugate gradient method. *Journal of Computational and Applied Mathematics*, 317:672–684, 2017.

[58] Žiga Povalej. Quasi-newton's method for multiobjective optimization. *Journal of Computational and Applied Mathematics*, 255:765–777, 2014.

Appendices

The neural network in Figure 19, which has one hidden layer, is designed to evaluate the effectiveness of the SQA algorithm in network pruning problems. All the edges between the input, hidden, and output layers will be pruned.

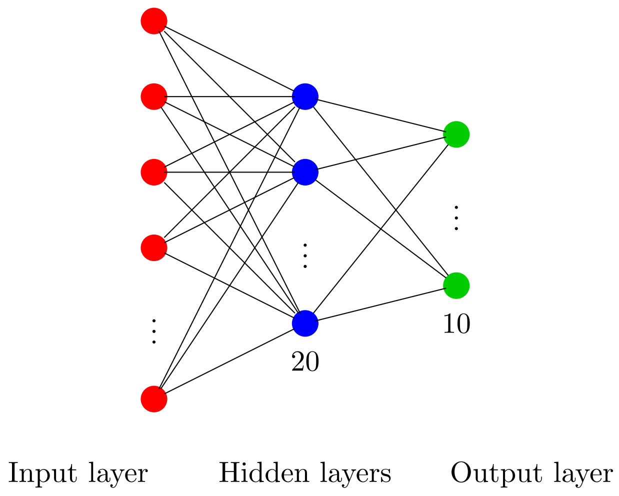


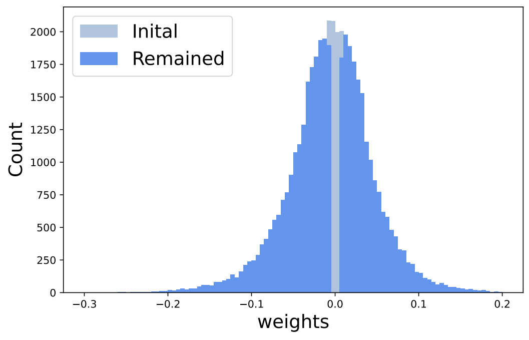
Figure 19: Structure of neural network with three hidden layers. This network is designed to evaluate the effectiveness of SA and SQA in network pruning.

Algorithm 1 Simulated Annealing

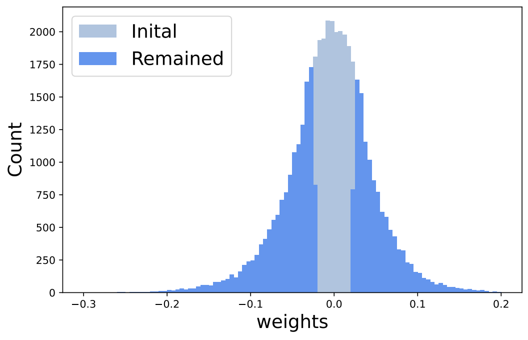
```
    Input: Neighborhood Rule, Objective function f(x)
    Output: Optimal Solution $x$
 1: $x$ = initial solution
 2: $T_0$ = initial temperature
 3: for iter = $1, 2, \ldots,$ MaxStep do (or other stopping rule)
 4:     $T$ = CoolingSchedule($T_0$, iter)
 5:     $x_{\text{new}}$ = NeighborhoodRule($x$)
 6:     $\Delta = f(x_{\text{new}}) - f(x)$
 7:     $r \sim$ U(0,1)
 8:     if $\Delta < 0$ then
 9:         $x = x_{\text{new}}$
10:     else if $r < AcceptProb(T, \Delta)$ then
11:         $x = x_{\text{new}}$
12:     end if
13: end for
14: return $x$
```

Algorithm 2 Simulated Quantum Annealing

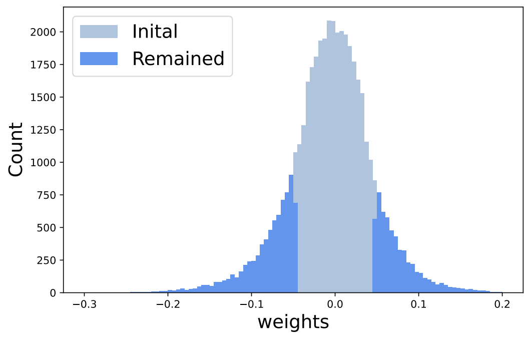
```
    **Input:** Neighborhood Rule, Hamiltonian function $\mathcal{H}(t)$, the number of Trotter slices $M$
    **Output:** Ground State $\sigma$
 1: $T_0, \mathit{\Gamma}_0$ = initial temperature, initial transverse field
 2: $\gamma_0 = \frac{T_0}{2}\ln\left(\coth\left(\frac{\mathit{\Gamma}_0}{MT}\right)\right)$
 3: for m $= 1, 2, \ldots, M$ do
 4:     for i $= 1, 2, \ldots, N$ do
 5:         $\sigma_i^m = \mathrm{Random}(1, -1)$
 6:     end for
 7:     $\mathcal{H}(m) = \sum_{(i,j)} \frac{J_{ij}}{M}\sigma_i^m\sigma_j^m + \sum_i \frac{h_i}{M}\sigma_i^m + \gamma_0 \sum_i \sigma_i^m\sigma_i^{m+1}$
 8: end for
 9: for iter $= 1, 2, \ldots,$ MaxStep do (or other stopping rule)
10:     $T = \mathrm{CoolingSchedule}(T_0, \text{iter})$
11:     $\mathit{\Gamma} = \mathrm{CoolingSchedule}(\mathit{\Gamma}_0, \text{iter})$
12:     $\gamma = \frac{T}{2}\ln\left(\coth\left(\frac{\mathit{\Gamma}}{MT}\right)\right)$.
13:     for m $= 1, 2, \ldots, M$ do
14:         Randomly Choose one $\sigma_i^m = \sigma_i^m * (-1)$
15:         $\mathcal{H}_{\text{new}}(m) = \sum_{(i,j)} \frac{J_{ij}}{M}\sigma_i^m\sigma_j^m + \sum_i \frac{h_i}{M}\sigma_i^m + \gamma \sum_i \sigma_i^m\sigma_i^{m+1}$
16:         $\Delta = \mathcal{H}_{\text{new}}(m) - \mathcal{H}(m)$
17:         $r \sim$ U(0,1)
18:         if $\Delta < 0$ then
19:             Accept the Swap
20:         else if $r < \mathit{AcceptProb}(T, \Delta)$ then
21:             Accept the Swap
22:         else
23:             Reject the Swap $\sigma_i^m = \sigma_i^m * (-1)$
24:         end if
25:     end for
26: end for
27: return $\sigma$
```



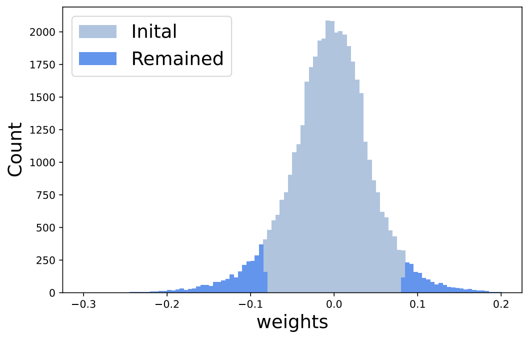
(a) 10% pruning



(b) 40% pruning

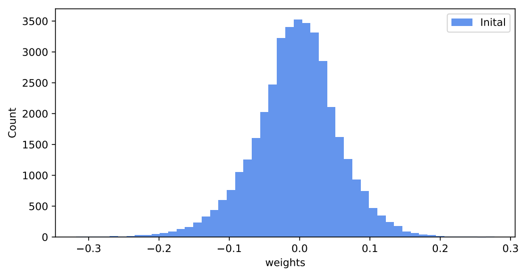


(c) 70% pruning

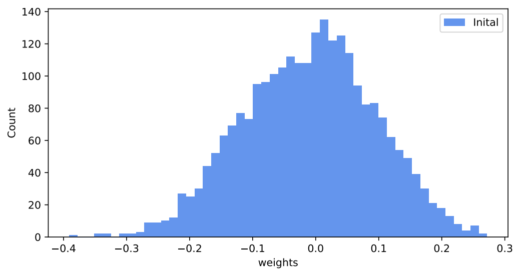


(d) 90% pruning

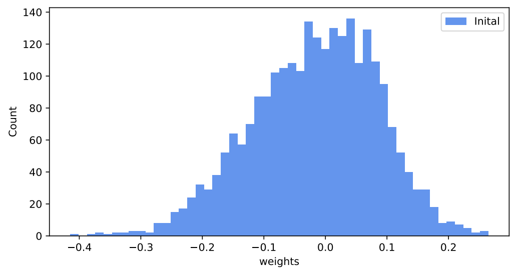
Figure 20: The histogram of the total weight parameters in the different pruning ratios using the min-K pruning method, the light blue represents the initial weight parameters without pruning, and the other represents the weight parameters after pruning, where 10% pruning in (a), 40% pruning in (b), 70% pruning in (c), and 90% pruning in (d).



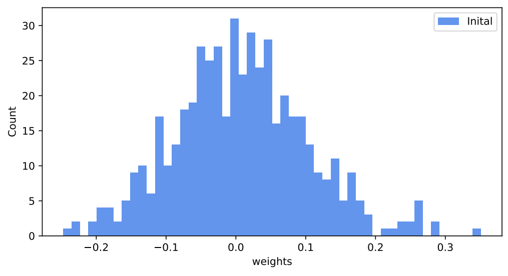
(a) The edges between input layer and hidden layer one.



(b) The edges between hidden layer one and hidden layer two.



(c) The edges between hidden layer two and hidden layer three.



(d) The edges between hidden layer three and output layer.

Figure 21: The histogram of the weight parameters between different layers. The nerual network shown in Figure 11 trained by MNIST dataset.