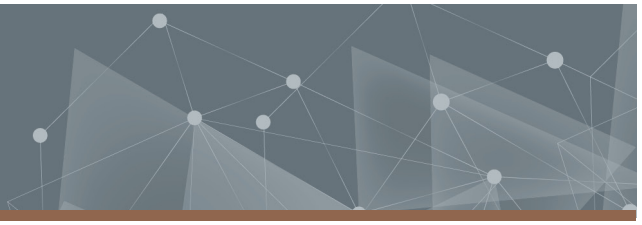
CHALMERS
UNIVERSITY OF TECHNOLOGY



Quantum approximate optimization using SWAP gates for mixing

Master's thesis in Complex Adaptive Systems

Carl Borgsten

DEPARTMENT OF PHYSICS

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Quantum approximate optimization using SWAP gates for mixing

Carl Borgsten

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Physics
Chalmers University of Technology
Gothenburg, Sweden 2021

Quantum approximate optimization using SWAP gates for mixing
Carl Borgsten

Supervisor: Mats Granath, Physics
Co-supervisor: David Fitzek, MC2
Examiner: Mats Granath, Physics

Master's Thesis 2021
Department of Physics
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Quantum approximate optimization using SWAP gates for mixing
Carl Borgsten
Department of Physics
Chalmers University of Technology

Abstract

In recent years, with the emergence of more complex optimization problems scientists have begun to look at how quantum computers can be utilized to solve big problems. One algorithm that can be used is the quantum approximation algorithm(QAOA). QAOA applies two different types of operators to a qubit string representing an initial state. One regulates the problem to be optimized and the other mixes the states such that all possible solutions to the problem are being tested. My work investigates the possibility to use SWAP gates for mixing instead of the ordinary Pauli X operator for the traveling salesman problem. The advantage of the SWAP mixer is that it swaps the values of two qubits in the qubit string compared to the Pauli X operator that changes the value of a single qubit. My results show that the SWAP mixer works well for the traveling salesman problem and has the potential to be used in more complex problems.

Keywords: quantum, computer, QAOA, qubit, TSP, SWAP, algorithm, optimize, mixing, state.

Acknowledgements

I would like to acknowledge the support I have received from my supervisor Mats Granath and my co-supervisor David Fitzek and for all the guidance and help during this work.

Carl Borgsten, Gothenburg, June 2021

Contents

List of Figures

1
Introduction

From the point computers were invented they have been used to solve equations and problems. With the development of better computers with more memory over time, bigger mathematical problems have been able to be solved. [1] With companies and scientists wanting to solve bigger and bigger problems the time it takes a computer to solve the problems increases a lot. To address this scientist has started to look at another type of computer to solve optimization problems. This is the quantum computer that uses quantum mechanics to solve optimization problems in a way that differs from an ordinary computer. [2] Ordinary computer stores information in bit strings consisting of either a 0 or a 1. In a quantum computer the information is also in a string, a quantum bit string or qubit string. Every qubit in this string can be 0 or 1 as an ordinary bit but it can also be a superposition of 0 and 1. This makes it potentially more powerful for calculating than an ordinary computer. [3] The quantum computer uses a quantum algorithm to solve an optimization problem. In this paper, I will look more at the Quantum Approximate Optimization Algorithm often called QAOA. As the name suggests it is an approximate algorithm which is not guaranteed to find the best solution. QAOA uses two types of operators on an initial state to find an solution. One that controls the problem that will be optimized and the other operator mixes the states such that all solutions will be compared. This operator is called the mixer and can be built in different ways. In the ordinary QAOA, the mixer is built by Pauli X operators, which changes the value of a single qubit. [4] However, in this paper, I will look at how the QAOA works with a Swap mixer, that instead swaps value between two qubits. This has been tested before with the Max-κ-Colorable-Subgraph problem where they call the swap mixer the ring mixer. [5] What will differ this paper from that work is that in this one I will look at the Traveling Salesman Problem instead and see if the swap mixer is a good solution in that case.

2
Theory

2.1 Quantum Computers

Quantum computers are built on two main principles, these are superposition and entanglement. With superposition means the attribute that an element can be in two states at the same time and the final state only known after a measurement of the element. Schroedinger's cat is an example of this where the knowledge of the cat's well-being is only known after the box is opened. This makes an advantage for the quantum computer compared to an ordinary computer. In an ordinary computer, the information is stored in a binary string of bits that are either 0 or 1. In the quantum computer, the information is stored in quantum bits or qubits that can be either 0, 1, or a superposition of both. This makes the qubit more powerful than the ordinary bit. [3]
The principle of entanglement means that the state of different elements correlate [3]. For example in the entangled state:

$$\frac{1}{\sqrt{2}}(|00> + |11>), \tag{2.1}$$

the probability of each states are $|\frac{1}{\sqrt{2}}| = 1/2$. However, if the first qubit are measured to be 0 the second qubit has 100% probability of being 0. The same can be said if the first qubit is measured to be 1.

2.2 QAOA

The quantum approximate optimization algorithm or in shorter form QAOA is an optimization algorithm used on quantum computers. The solutions to the problems are represented as states. These states can be represented by a string of qubits where each qubit can have a value of 0 or 1. When the qubit value is 1 the qubit is part of the solution and when the qubit valuen is 0 the qubit is not in solution. In the QAOA you start with preparing an initial state. This state can be a single state or a superposition of n states z [4]:

$$s = \frac{1}{\sqrt{n}} \sum_{z} |z> \tag{2.2}$$

The initial state is then affected by some operators to find an approximate solution that optimizes the problem. These operators are of two different kinds $U(\gamma, H)$

and $U(\beta, M)$. These operators are unitary which are the type of operators used in a quantum computer. $U(\gamma, H)$ regulates the optimization problem and applies a phase to the state. This operator consists of the Hamiltonian of the problem and an angle γ. [4]

$$U(\gamma, H) = e^{-i\gamma H} \tag{2.3}$$

$U(\beta, M)$ is used to mix the states in order for all states to be tested. This is done by a mixer M and an angle β.

$$U(\beta, M) = e^{-i\beta M} \tag{2.4}$$

These gates are applied to the initial state in several consecutive layers p. The higher number of layers increases the performance of the algorithm because of the more parameters used, but also require more operations in the quantum computer. Each layer has two parameters, these are β_i and γ_i with i representing the layer of the parameters. With this the expression of the algorithm can be written:

$$\begin{aligned} |\gamma, \beta> = & U(\gamma_p, H)U(\beta_p, M)...U(\gamma_1, H)U(\beta_1, M)|s_0> = \\ & e^{-i\gamma_p H}e^{-i\beta_p SWAP}....e^{-i\gamma_1 H}e^{-i\beta_1 M}|s_0> \end{aligned} \tag{2.5}$$

In this expression s_0 represents the initial state of the problem, $\gamma_1, ..., \gamma_p$ and $\beta_1, ..., \beta_p$ represents the parameters for each layer, H is the Hamiltonian of the problem we want to optimize and M is the mixer that mixes the states [4]. In this paper, the mixer is the swap gate that swaps two qubits instead of the ordinary Pauli X operator that swaps the probability of $|0>$ and $|1>$ for every qubit in the string. This swap gate can be described in matrix form as[6]:

$$SWAP = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

The SWAP operator is both Hermitian and unitary such that $(SWAP)^2 = 1$ [7]. This means the operator can be written in terms of cosine and sine, this gives the SWAP mixer:

$$\begin{aligned} e^{-i\beta SWAP} = & \cos(\beta) * I + i * \sin(\beta)SWAP = \\ & \begin{bmatrix} \cos(\beta) + i\sin(\beta) & 0 & 0 & 0 \\ 0 & \cos(\beta) & i\sin(\beta) & 0 \\ 0 & i\sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & \cos(\beta) + i\sin(\beta) \end{bmatrix} \end{aligned} \tag{2.7}$$

Every state's performance can be described by an energy. A lower energy represents a better solution to the optimization problem. This energy can be calculated with the expectation value of the Hamiltonian in the current state:

$$E = < \gamma, \beta|H|\gamma, \beta >, \tag{2.8}$$

with H again representing the Hamiltonian and $|\gamma, \beta>$ being the state. This means that some parameters lead to a solution with lower energy and the goal is to find the $2p$ parameters that finds the optimal solution, [4] in other words the parameters that gets the lowest expectation value and represent the best solution:

$$E_{min}(\gamma,\beta) = \sum_{z} < s_0|U(\gamma_1,H)^\dagger\, U(\beta_1,M)^\dagger \ldots U(\gamma_p,H)^\dagger\, U(\beta_p,M)^\dagger\, H \\ U(\gamma_p,H)\, U(\beta_p,M) \ldots U(\gamma_1,H)\, U(\beta_1,M)\, |s_0> \tag{2.9}$$

2.3 Traveling Salesman Problem

The Traveling Salesman Problem or TSP is an optimization problem to find the shortest path of a traveling salesman who wants to visit some cities and wants to find the shortest path while only visiting every city once before returning to the starting city. This is an NP-hard problem meaning that it falls in a class of computationally difficult problems to solve and therefore is a good test for the QAOA. There is some classical algorithms to solve the TSP most notably the brute force variant where every possible solutions are compared. The number of solutions increases as $n_{cities}!$ resulting in a lot of solutions to be compared. Another method to solve TSP is to visit the nearest city for every stop. However, this will not always result in a good solution. [8] Other ways to solve the TSP is to use a stochastic optimization algorithm such as the Ant Colony Optimization Algorithm (ACO), the Particle Swarm Optimization Algorithm (PSO) or a Genetic Algorithm (GA). ACO are built in a similar way to how ants localizes food and resources. Ants lay down pheromone trails after they have found food. Other ants have a higher probability of choosing a trail with a higher pheromone concentration. The algorithm are built the same way, with different agents traversing the grid with every edge having different probabilities. These probabilities depends on the number of agents that has traversed that edge. PSO on the other hand uses the behavior of birds and their social behaviour. The different agents searches the space with different velocities based on their path. This path are compared to the agents previously best path and the swarms best path. With this a solution to the problem can be achieved. The last algorithm GA are based on biology. A number of chromosomes are generated, with every chromosome having a fitness value. In optimization problems where you want to minimize something a lower fitness value represent a better solution. For a number of generations chromosomes are removed, the chromosomes with a lower fitness value have a lower chance to be terminated. In every generation the chromosomes are exposed to crossover and mutations. When the best fitness value does not change the algorithm are finished. [9]
In this problem we look at toy problems where the QAOA will be slower than a classical algorithm. To run QAOA of the TSP we need to find the Hamiltonian of the problem. We divide the Hamiltonian into three parts, $H = H_A + H_B + H_C$, one that fulfills the constraint that we visit every city, one that fulfills the constraint that we each time step go to a new city, and one that uses the traveled distance. First, we need to define a variable, $X_{i,\alpha}$ that is 1 if the city i is visited at time step α and zero otherwise. A solution to the TSP with this notation can be seen in figure

2.1. This is a suitable formulation for a quantum computer.

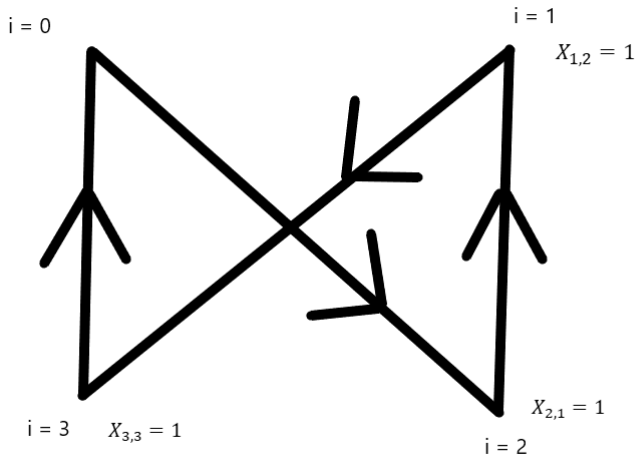


Figure 2.1: Figure over how a solution to the TSP can be described with the variable $X_{i,\alpha}$.

With this variable the constraints can be written as:

$$\sum_i X_{i,\alpha} = 1 \ \forall \alpha \tag{2.10}$$

$$\sum_\alpha X_{i,\alpha} = 1 \ \forall i \tag{2.11}$$

From these constraints the Hamiltonians H_A and H_B can be expressed as:

$$H_A = \sum_i \left(1 - \sum_\alpha X_{i,\alpha}\right)^2 \tag{2.12}$$

$$H_B = \sum_\alpha \left(1 - \sum_i X_{i,\alpha}\right)^2 \tag{2.13}$$

With these expressions, a state that visits a city twice or two cities at the same time step will receive a penalty resulting in higher energy. This means the states that fulfill the constraints will therefore end up with lower energy.
Before H_C is defined the problem can be simplified. In the ordinary TSP, the starting city is randomized. This means that to solve the TSP n^2_{cities} qubits are needed to find the solution. However, if the starting city is fixed the needed number of qubits is instead $(n_{cities} - 1)^2$ which makes a big difference computationally for the small problems we consider. Proof of this is that the 4 city path $1-2-3-4-1$ represents the same solution as $3-4-1-2-3$ and with that, the first city can be fixed. With

this simplification, the cost part of the Hamiltonian can be defined as:

$$H_C = \sum_{i=1}^{n_{cities}-1} \sum_{j=1}^{n_{cities}-1} c_{i,j} \sum_{\alpha=1}^{n_{cities}-2} X_{i,\alpha} X_{j,\alpha+1} + \sum_{i=1}^{n_{cities}-1} c_{0,i} X_{i,1} + \sum_{i=1}^{n_{cities}-1} c_{i,0} X_{i,0} \tag{2.14}$$

In equation (2.14), c is a matrix of the distances between every city. The first part of the expression represents the distances between every city except the starting city. The second part represents the distance from starting city to the first visited city while the last expression represents the distance from the last visited city to the starting city. The edge between the last city visited and the start city $i = 0$ are defined with $\alpha = 0$.

2.4 Create SWAP gate

The SWAP changes the value of two qubits. This means that the Hamming distance to $|000..>$ are conserved. This can be utilized to limit the space searched. How this is done are described below. With the X notation, a viable solution that does not break the constraint can be written as a matrix with a row sum of 1 and a column sum of 1. With each column being the timestep 1, 2, and 3 and the rows being city 1, 2, and 3. For example, a good solution of the 9 qubit case can be:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

this correspond to the first city being visited at timestep 1, the second city at timestep 2 and the third city at timestep 3. A bad solution can be:

$$X = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \tag{2.16}$$

that visits all the cities at the same timestep. With this fact we can see that the good solutions will have n_{stops} ones, with each one being in a different row and column as the other ones for a total of $n_{stops}!$ good solutions. There are $n_{stops}^{n_{stops}}$ possible solutions visiting all the cities but that may be on the same timestep another city is visited or visited multiple times. This means the initial state can be modified to only look at the good solution by starting in a state with ones in different rows and columns and then swap two columns. For example in the 9 qubit case a swapper that would swap column 1 and 2 can look like this:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

Starting in a viable state and swapping two columns end up in a new viable state. However, to do this a gate of the size n_{stops}^2 has to be implemented which is tricky when n_{stops} increases. Instead, a simpler swap mixer can be used that takes more steps to achieve the same thing. This can be done in two ways, the first is to swap the timestep each city is visited by moving the ones in each row and skip swapping between rows. This means that if only states with only one 1 in each row are used as the initial state the resulting states of the QAOA should fulfill this property as well. If the starting state is the same as above this can look like this:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

And the same result is achieved after one more step. The other way is very similar to the first, but instead of swapping which timestep every city is visited the city visited at every timestep is swapped by moving the ones in each column and not in the rows. This would look like this:

$$X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \rightarrow X = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.19}$$

The first simpler swap mixer will be used in this paper. In this case, the qubits that are swapped can be described by the expression below with i representing the city and j the timestep:

$$Q_{SWAP} = \sum_{\alpha=1}^{n_{cities}-1} \sum_{j=\alpha+1}^{n_{cities}-1} SWAP_{(i,\alpha),(i,j)} \ \forall i \tag{2.20}$$

In the above expression the second sum is from $i+1$ because a swap between qubit 0 and 1 is the same as a swap between qubit 1 and 0 and is unnecessary to do twice. With this SWAP mixer the searched space will be a little bit bigger than for the optimal SWAP mixer that swaps columns as described earlier. However, compared to the single qubit bitflips in the standard QAOA the searched space will be smaller. For the single qubit bitflips the constraints are not preserved as it explores the full $2^{n_q ubits}$ space. This makes it much more difficult to find viable solutions and minimize the energy and path length. [7]

3
Methods

3.1 Simulation of quantum computer

Although quantum computers of increasing size are being constructed, access is limited. They are also prone to noise and have limited gate sets, making simulations difficult. Some initial work using QAOA on real quantum computers has been done. [10] Instead, in this paper, the quantum computer will be simulated. A quantum computer can be simulated in a few steps. First, an initial state has to be prepared, next apply operators to the state, and finally compute the final state. All this can be done with the python package Qiskit. In Qiskit different quantum circuits can be created to imitate a quantum computer. These quantum circuits can be built to represent the expression in equation (2.5) with all the gates. This quantum circuit can then be executed with a certain backend that computes the final state [11].
First, the initial state has to be applied to the quantum circuit then each layer of operators from equation (2.5) can be added. Every layer has a certain γ_i and β_i that are being optimized. In this paper, the state vector backend is used. This backend returns the amplitude of every state. [12] With this, the expectation value of the energy for the current state can be calculated by multiplying with the Hamiltonian. The total probability of all the states are 1, this means that the state with the highest amplitude is the most probable state measured if one would run the circuit on a quantum computer.
To find the optimal angles the parameters are optimized with the differential evolution function in the library SciPy in python. Differential evolution is a stochastic optimizer that creates several solutions. Two such solutions are then mixed and if it is better than the original the original is replaced. [13] Basin hopping was another optimization algorithm that was tested but was less efficient and took much more time to run. To use the gates in Qiskit, the variable $X_{i,\alpha}$ has to be an operator. This can easily be fixed with a variable substitution to the Pauli Z operator:

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \tag{3.1}$$

Making the substitution.

$$X_{i,\alpha} = \tfrac{1}{2}(1 - Z_{i,\alpha}), \tag{3.2}$$

this becomes 0 when $Z_{i,\alpha}$ is $|0>$ and 1 when $Z_{i,\alpha}$ is $|1>$ [14].
In the simulation in Qiskit, the qubits are in a string in comparison with the matrix presented earlier. To solve this the matrix is flatten meaning in the 9 qubit case the first 3 qubits are the first row which represents when the first city is visited, the 3

next qubits are the second row and the last 3 qubits are the third row. This means the Hamiltonian can be rewritten as:

$$H_A = \sum_{i=1}^{n_{stops}} \left(1 - \sum_{\alpha=1}^{n_{stops}} X_{i,\alpha}\right)^2 = \sum_{i=1}^{n_{stops}} \left(1 - \sum_{\alpha=1}^{n_{stops}} \tfrac{1}{2}(1 - Z_{i,\alpha})\right)^2 = \sum_{i=1}^{n_{stops}} \left(1 - \sum_{\alpha=1}^{n_{stops}} \tfrac{1}{2}(1 - Z_{i*n_{stops}+\alpha})\right)^2 \tag{3.3}$$

$$H_B = \sum_{\alpha=1}^{n_{stops}} \left(1 - \sum_{i=1}^{n_{stops}} X_{i,\alpha}\right)^2 = \sum_{\alpha=1}^{n_{stops}} \left(1 - \sum_{i=1}^{n_{stops}} \tfrac{1}{2}(1 - Z_{i,\alpha})\right)^2 = \sum_{\alpha=1}^{n_{stops}} \left(1 - \sum_{i=1}^{n_{stops}} \tfrac{1}{2}(1 - Z_{i*n_{stops}+\alpha})\right)^2 \tag{3.4}$$

$$\begin{aligned} H_C = &\sum_{i=1}^{n_{stops}} \sum_{j=1}^{n_{stops}} c_{i,j} \sum_{\alpha=1}^{n_{stops}-1} X_{i,\alpha} X_{j,\alpha+1} + \sum_{i=1}^{n_{stops}} c_{0,i} X_{i,1} + \sum_{i=1}^{n_{stops}} c_{i,0} X_{i,0} = \\ &\sum_{i=1}^{n_{stops}} \sum_{j=1}^{n_{stops}} c_{i,j} \sum_{\alpha=1}^{n_{stops}-1} \tfrac{1}{2}(1 - Z_{i,\alpha})\tfrac{1}{2}(1 - Z_{j,\alpha+1}) + \sum_{i=1}^{n_{stops}} c_{0,i} \tfrac{1}{2}(1 - Z_{i,1}) + \\ &\sum_{i=1}^{n_{stops}} c_{i,0} \tfrac{1}{2}(1 - Z_{i,0}) = \\ &\sum_{i=1}^{n_{stops}} \sum_{j=1}^{n_{stops}} c_{i,j} \sum_{\alpha=1}^{n_{stops}-1} \tfrac{1}{2}(1 - Z_{n_{stops}*i+\alpha})\tfrac{1}{2}(1 - Z_{n_{stops}*j+\alpha+1}) + \\ &\sum_{i=1}^{n_{stops}} c_{0,i} \tfrac{1}{2}(1 - Z_{n_{stops}*i+1}) + \sum_{i=1}^{n_{stops}} c_{i,0} \tfrac{1}{2}(1 - Z_{n_{stops}*i}) \end{aligned} \tag{3.5}$$

With these modifications of the problem Hamiltonian, the SWAP mixer are ready to be tested.
The final version of the quantum approximate optimization algorithm used for the travelling salesman problem in this paper can be described in algorithm 1:

Algorithm 1 QAOA

1: Define the Hamiltonian $H = H_A + H_B + H_C$ as in equation (3.3), equation (3.4) and equation (3.5)
2: Define the mixer as in the equations (2.7), (2.20)
3: Define an initial state s_0
4: Construct circuits for the operators $U(\gamma, H)$ and $U(\beta, M)$ from equation (2.3) and equation (2.4)
5: Choose the amount of layers p and create circuit for the whole algorithm as in equation (2.5)
6: Optimize the parameters γ_i and β_i
7: Get approximate solution to the problem with these optimal parameters

3.1.1 Prepare initial state

The first two steps in algorithm 1 has already been done earlier. However, in the third step the initial state has to be defined and prepared. A quantum computer start in the state $|000...>$. If the initial state should be a superposition of all the possible states the Hadamard gate can be used. To get a specific set of states, different gates can be applied to the start state $|000...>$. However, Qiskit has a function that given a list of the vector we want to start in, as well as the qubits to initialize in this state construct the initial state we want. [11]

3.1.2 Construct circuits for operator

After the initial state has been prepared the different operators has to be applied to this state. The quantum circuit for the operator $U(\gamma, H)$ can be described with the following for loops. The first describes the terms in the Hamiltonian with a single Pauli Z operator. The second loop describes the terms in the Hamiltonian with a product of Pauli Z operators:

3.1.2.1 Single qubit operator

```
for qubits in Hamiltonian_single:
    qubit = int(qubits[0])
    weight = qubits[1]
    qc.rz(2 * gamma * weight, qubit)
}
```

3.1.2.2 Double qubit operator

```
for qubitPair in Hamiltonian_double:
    qubit1 = int(qubitPair[0])
    qubit2 = int(qubitPair[1])
    weight = qubitPair[2]
    qc.rzz(2 * gamma * weight, qubit1, qubit2)
}
```

In these for loops gamma is the angle applied to layer i, weight are a constant multiplied with every Pauli Z operator in the Hamiltonian. qubit, qubit1 and qubit2 are the qubits the gate are applied to. rz is a single qubit rotation around the Z axis described as

$$rz(\lambda) = e^{-i\frac{\lambda}{2}Z},$$

for an angle λ that are divided by two. This is why the angle gamma aer multiplied with 2. rzz is a 2 qubit interaction described as

$$rzz(\theta) = e^{-i\frac{\theta}{2}Z\oplus Z},$$

for an angle θ. [11]
In the case for the SWAP mixer, a unitary operator were created as in equation (2.7).

3.1.2.3 SWAP unitary operator

```
val1 = complex(np.cos(beta), np.sin(beta))
val2 = complex(0, np.sin(beta))
val3 = complex(np.cos(beta), 0)


swap_mixer = Operator([[val1, 0, 0, 0],
                       [0, val3, val2, 0],
                       [0, val2, val3, 0],
                       [0, 0, 0, val1]])
}
```

This SWAP unitary operator were then used for all the qubits in equation (2.20).

3.1.2.4 SWAP qubits

```
for qubitPair in SWAP_qubits:
    qubit1 = qubitPair[0]
    qubit2 = qubitPair[1]
    qc.unitary(swap_mixer, [qubit1, qubit2], label = 'swapmixer')
}
```

Now with these quantum circuit defined they can be used in layers p to create a final quantum circuit as in equation (2.5). With the final quantum circuit the parameters γ_i and β_i can be optimized with differential evolution and an approximate solution to the problem can be achieved.

4

Results

4.1 Performance of SWAP mixer

To test the performance of the SWAP mixer a few compositions of the initial state will be used. This test will have 9 qubits representing 4 cities with 3 stops where city 0 is the starting city. All the states will have properties as described earlier to only swap between states consisting of 3 stops. The different compositions of the initial state will be only 1 bad state, only 1 good state, superposition of all bad states, superposition of all good states, and finally a superposition of all states. A bad state is in this case a state that as in equation (2.16) has more than one 1 in a column and a good state has only one 1 in every row and column as in equation (2.15). In the 9 qubit case this gives 6 good states and 21 bad states for a total of 27 states. This is a large reduction due to using SWAP mixer, compared to the $2^9 = 512$ states that would be explored with the standard bit-flip mixer. Results for $p = 1$ to $p = 10$ layers will be documented with a predicted better performance for higher p. All of the trials use differential evolution to find optimal parameters. Every state has an energy calculated as in equation (2.8) with its corresponding state vector and the Hamiltonian. This energy will be lower for the states that satisfy the constraints and represent a shorter path. The best path in this small problem can be seen in figure 4.1. Note that traversing the path the other way around results in the same distance and therefore has the same energy. These paths are represented by the states $|001100010>$ and $|100001010>$. In the following figures the states representing the best paths will be showed as a red bar.

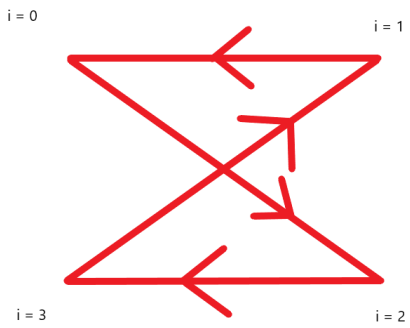


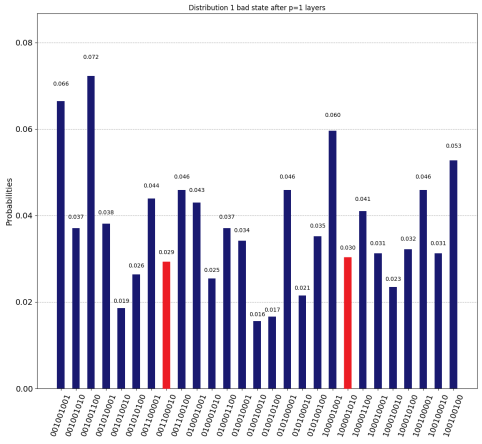
Figure 4.1: Figure showing the path with the lowest energy in the toy problem. The reverse path shares the lowest energy.

4.1.1 Starting in one state that violates the constraints

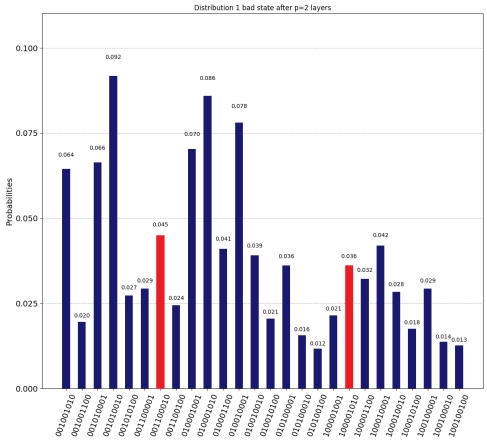
When starting in only one state the initial state from equation (2.2) will be:

$$s_0 = |001001001>, \tag{4.1}$$

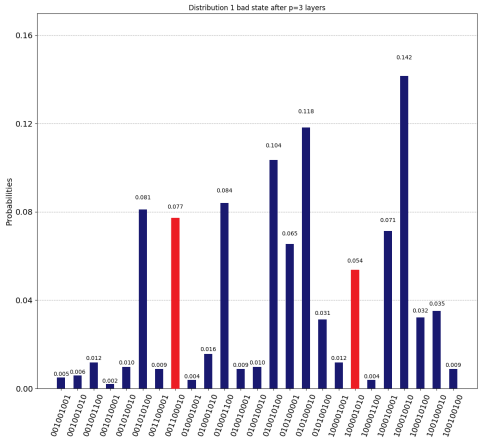
The first test start in the initial state $|001001001>$, this represents the path visiting all cities at the third time step. In the following figures the probability of all the states after p layers can be seen.



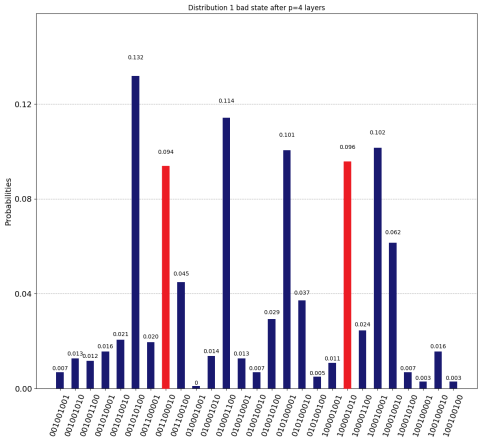
(a) Probabilities for the states after p=1 layers.



(b) Probabilities for the states after p=2 layers.

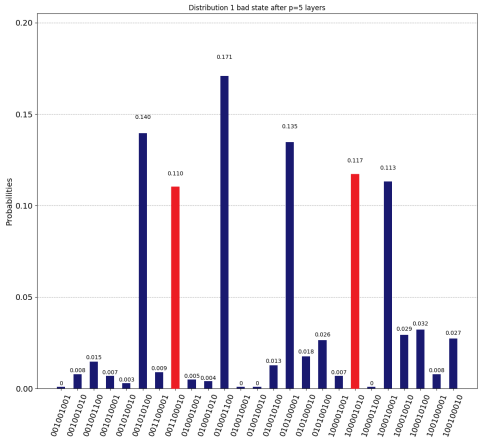


(c) Probabilities for the states after p=3 layers.

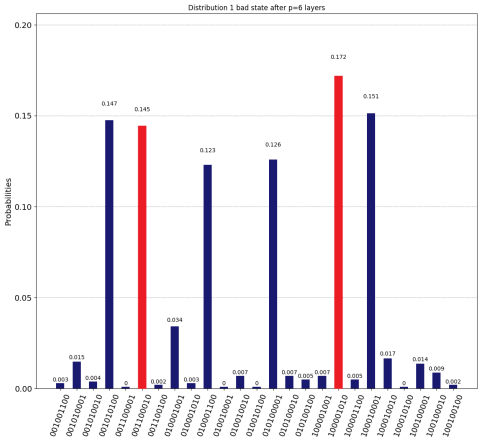


(d) Probabilities for the states after p=4 layers.

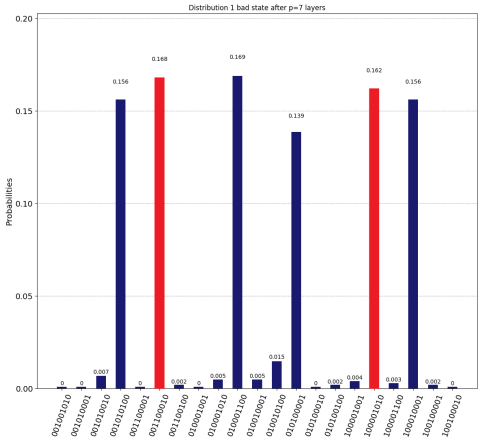
Figure 4.2: Probabilities for the different states after p layers for the initial state $s_0 = |001001001>$. Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



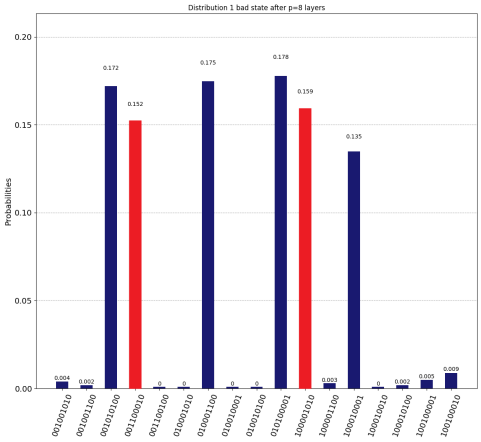
(a) Probabilities for the states after p=5 layers.



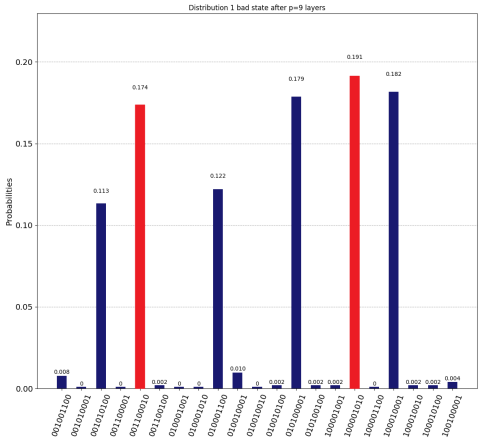
(b) Probabilities for the states after p=6 layers.



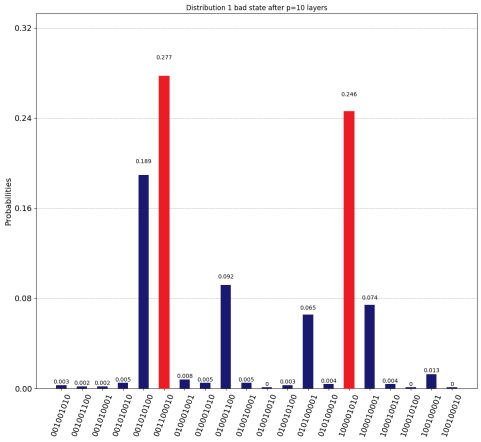
(c) Probabilities for the states after p=7 layers.



(d) Probabilities for the states after p=8 layers.



(e) Probabilities for the states after p=9 layers.



(f) Probabilities for the states after p=10 layers.

Figure 4.3: Probabilities for the states after p layers for the initial state $s_0 = |001001001>$. Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier

All these distributions results in different expectation values for the energy. How this value changed through each layer can be seen in figure 4.4. This shows a comparison to the lowest energy of the best state. The numbers are smaller than 1 because the lowest energies are negative.

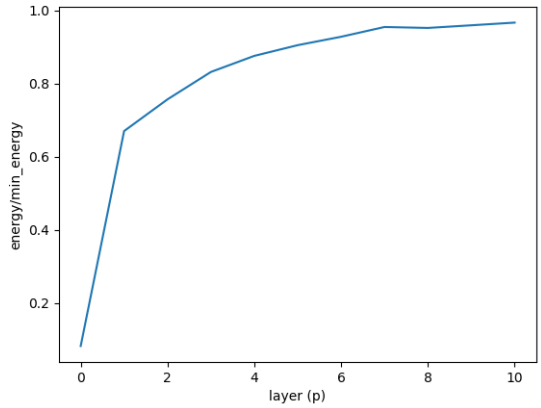


Figure 4.4: Evolution of the expectation value of the energy calculated as in equation 2.8 for each layer, compared to the energy of the best state when starting in a bad state.

A distribution of all the 27 possible states appears after the first layer of operations in figure 4.2a when starting in the bad state $s_0 = |001001001>$. This can be compared to an initial state of all the possible states but with every state having a different probability in the beginning, running for only $p = 9$ layers instead of $p = 10$. With the increase of layers p the probabilities of the good states increases to end up around 1/6 for every good state in figure 4.3c and figure 4.3d.

4.1.2 Starting in one state that satisfy the constraints

The SWAP mixer worked pretty good for an initial state consisting of a bad state but what happens if the initial state is a good state?

In this test the performance when starting in one good state are examined. As when starting in only one bad state the probability of the starting state will be 1. The initial state for this test is $|010001100>$ representing the path 0-3-1-2-0, with 0 being the starting city.

The following figure describes how the distribution of the states have changed after p = 1 layer.

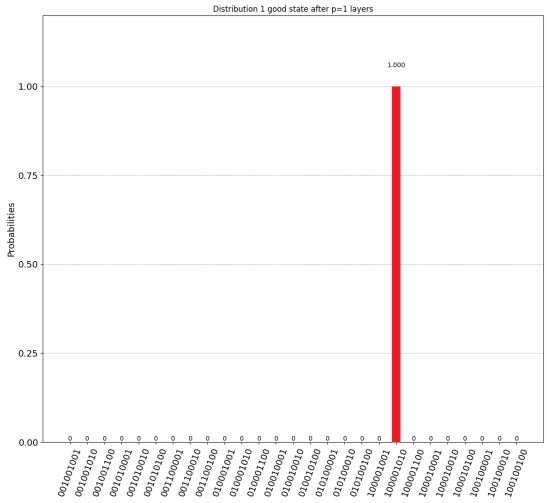


Figure 4.5: Probabilities for the states after p = 1 layer for the initial state $s_0 = |010001100>$.

All these distributions results in different expectation values for the energy. How this value changed with each layer can be seen in figure 4.6.

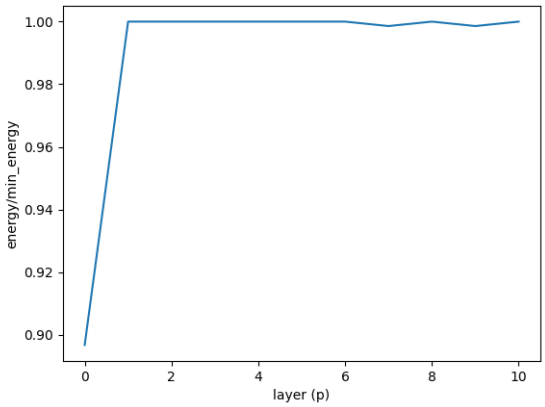


Figure 4.6: Evolution of the expectation value of energy calculated as in equation 2.8 for each layer, compared to the energy of the best state when starting in a good state.

In figure 4.5 the state has already changed from the initial state $s_0 = |010001100>$ to the state $|100001010>$ after the first layer. The probability of the other states are 0. For higher number of layers $p > 1$ I got the same probabilities of the final state with a 100% probability of ending up in $|100001010>$. This is also the best state overall with the lowest energy which can be seen by its red color. This can also be seen in figure 4.6 where the expectation value of the energy instantly increases to the maximum. The states differs by only 4 qubits making it quite simple to SWAP into. However, the resulting state is $|100001010>$ instead of the other best state

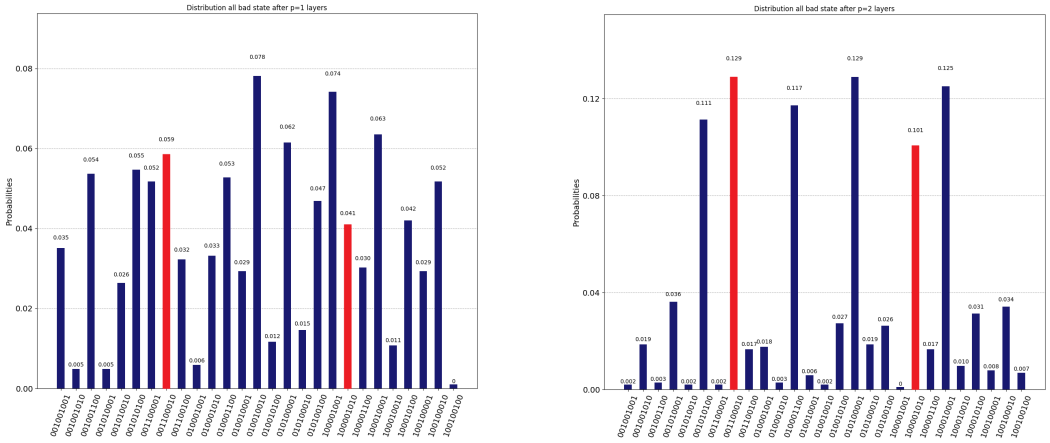
$|001100010>$. The difference between those are that the other state differs by 6 qubits compared to the initial state.
Even though the single initial state worked in this case the advantages of a quantum computer with the superposition were unutilized. This solution to the problem can be compared to a classical algorithm with a start in a good state and then move around the position of the 1:s to create permutations of the original state. This is done until the best state is found. Instead how the SWAP works with an initial state consisting of a superposition of multiple states is more interesting.

4.1.3 Starting in a superposition of all the states that violates the constraints

In the first superposition test, an initial state of all the bad states are tested. The number of bad states are 21 as stated earlier which means that the initial state are represented by:

$$s_0 = \frac{1}{\sqrt{21}}(|001001001> + |001001010> + |001001100> + |001010001> + |001010010> + |001100001> + |001100100> + |010001001> + |010001010> + |010010001> + |010010010> + |010010100> + |010100010> + |010100100> + |100001001> + |100001100> + |100010010> + |100010100> + |100100001> + |100100010> + |100100100>) \quad (4.2)$$

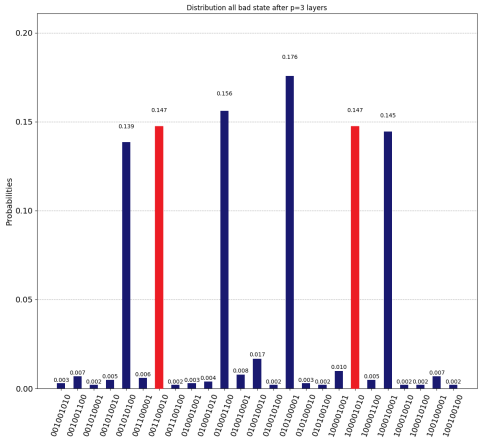
The following figures describes how the distribution of the states change with each layer p.



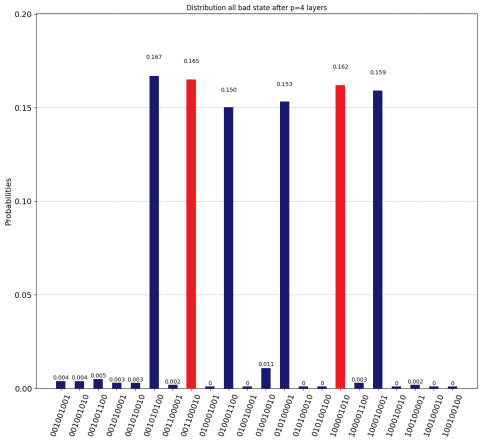
(a) Probabilities for the states after p=1 layers.

(b) Probabilities for the states after p=2 layers.

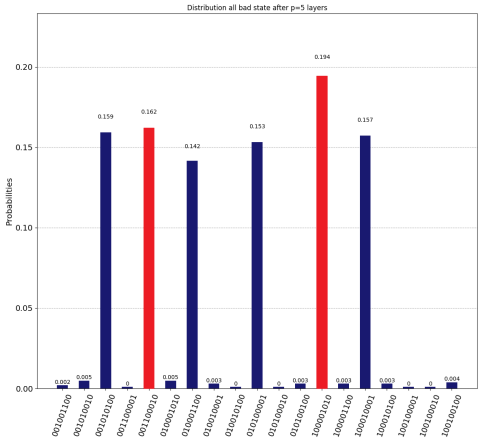
Figure 4.7: Probabilities for the different states after p layers for the initial state described by equation (4.2). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



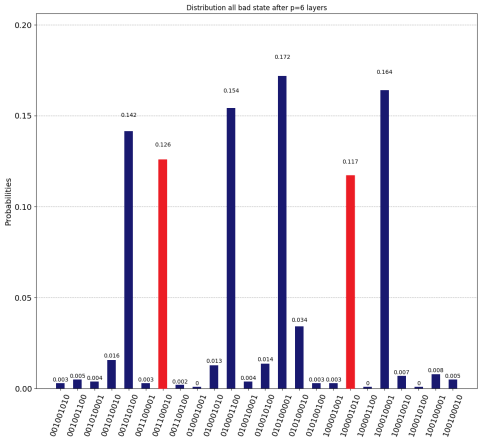
(a) Probabilities for the states after p=3 layers.



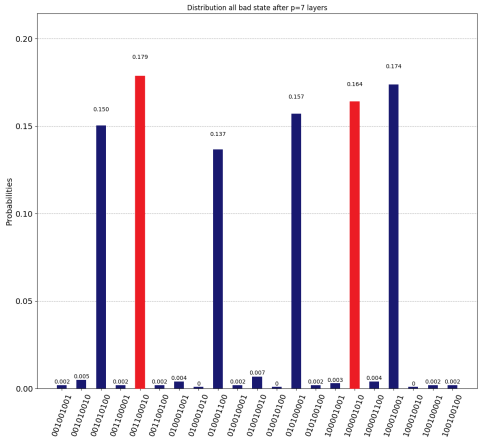
(b) Probabilities for the states after p=4 layers.



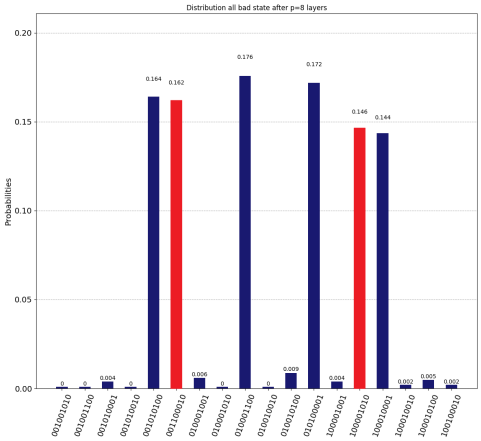
(c) Probabilities for the states after p=5 layers.



(d) Probabilities for the states after p=6 layers.

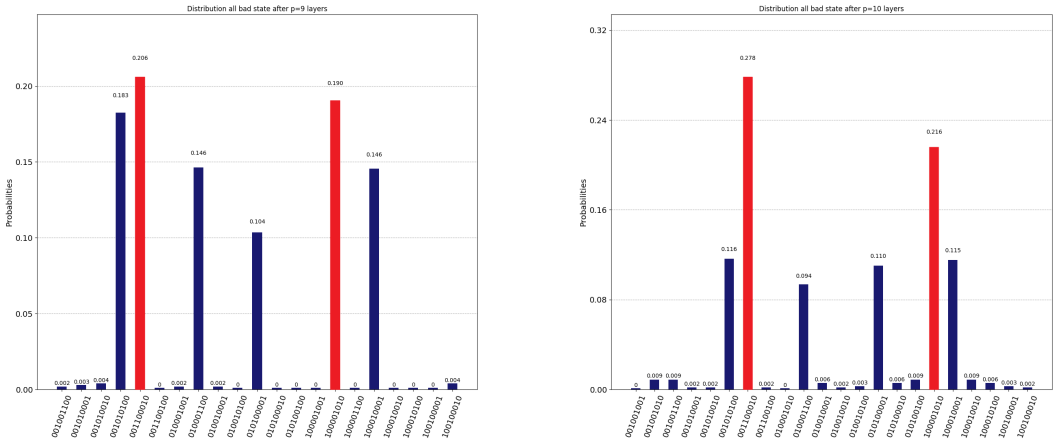


(e) Probabilities for the states after p=7 layers.



(f) Probabilities for the states after p=8 layers.

Figure 4.8: Probabilities for the different states after p layers for the initial state described by equation (4.2). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



(a) Probabilities for the states after p=9 layers.

(b) Probabilities for the states after p=10 layers.

Figure 4.9: Probabilities for the different states after p layers for the initial state described by equation (4.2). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.

All these distributions results in different expectation values of the energy. How the energy changed with each layer can be seen in figure 4.10.

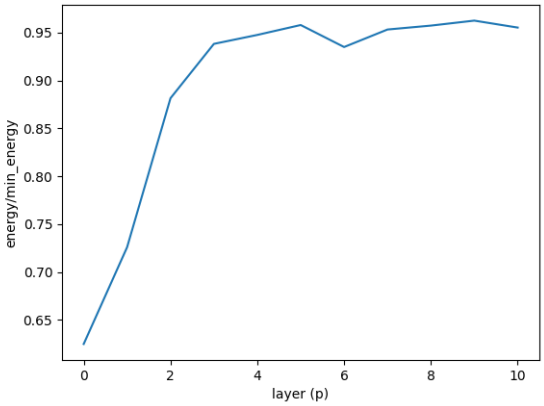


Figure 4.10: Evolution of the expectation value of energy calculated as in equation 2.8 for each layer, compared to the energy of the best state when starting in a superposition of all bad states.

The superposition of multiple bad states has pretty much the same development as the single bad state. First changing to a state of all the good states with a probability of 1/6 for all the states after $p = 8$ layers in figure 4.8f. Before ending up with a higher probability of the states $|001100010>$ and $|100001010>$ that have the lowest energy.

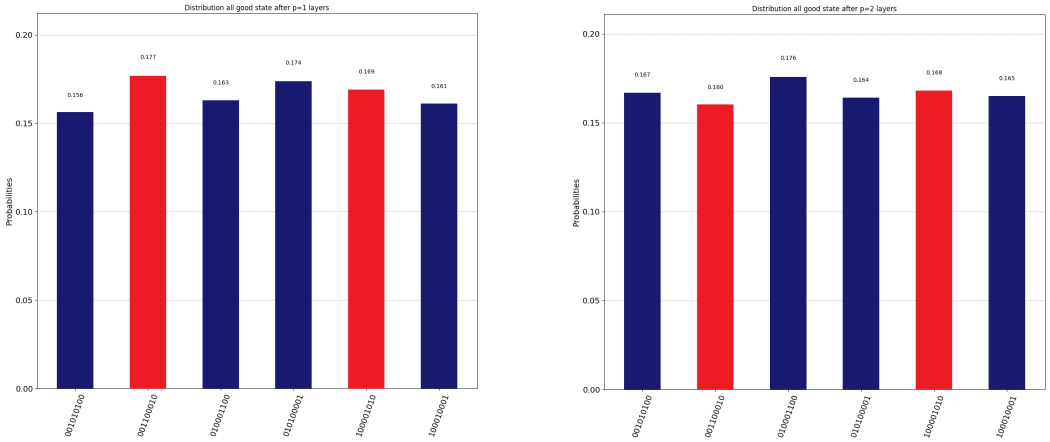
4.1.4 Starting in a superposition of all the states that satisfy the constraints

The superposition of only bad states worked pretty much at the same level as the single bad state. How does the superposition of all good state compare to the single good state?

This test of the performance of the SWAP mixer for a superposition of all the good states consists of 6 good states described by the following expression:

$$s_0 = \frac{1}{\sqrt{6}}(|001010100> + |001100010> + |010001100> + |010100001> + |100001010> + |100010001>) \tag{4.3}$$

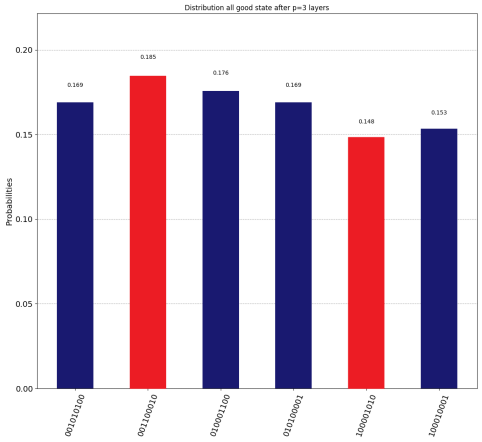
The following figures describes how the distribution of the states change with each layer p.



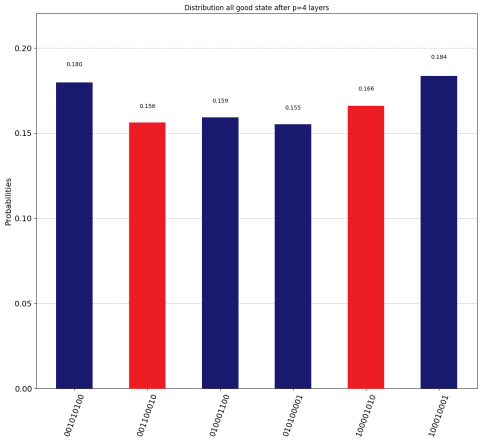
(a) Probabilities for the states after p=1 layers.

(b) Probabilities for the states after p=2 layers.

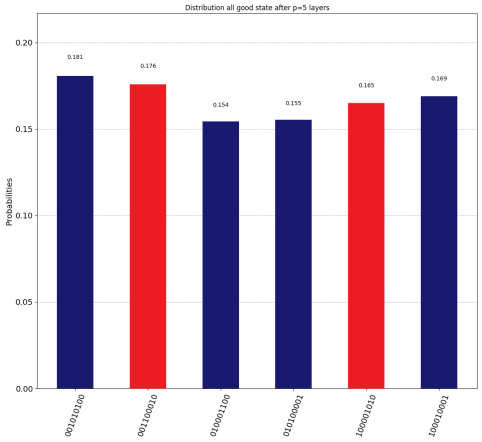
Figure 4.11: Probabilities for the different states after p layers for the initial state described by equation (4.3). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



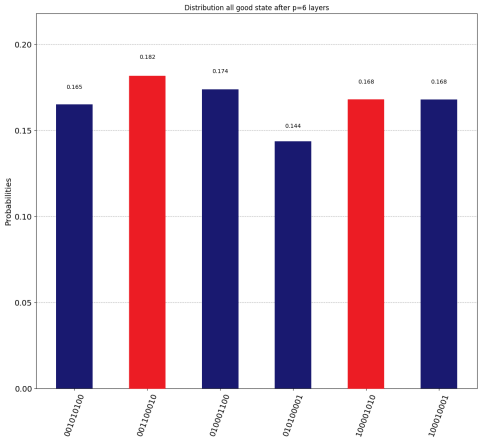
(a) Probabilities for the states after p=3 layers.



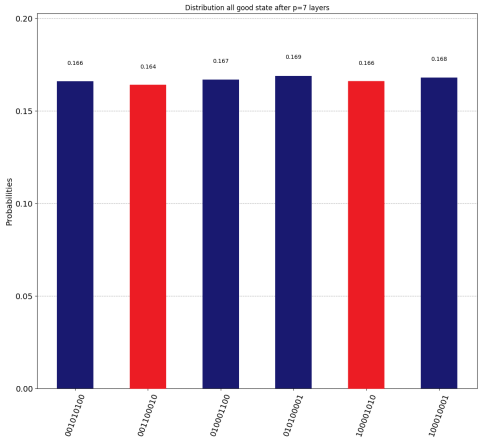
(b) Probabilities for the states after p=4 layers.



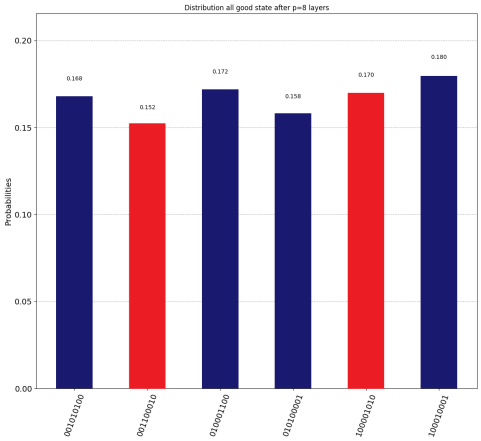
(c) Probabilities for the states after p=5 layers.



(d) Probabilities for the states after p=6 layers.

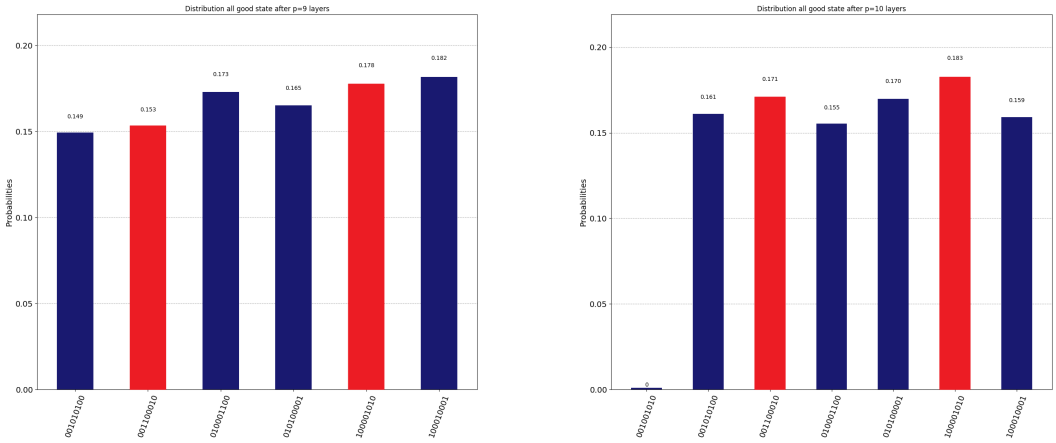


(e) Probabilities for the states after p=7 layers.



(f) Probabilities for the states after p=8 layers.

Figure 4.12: Probabilities for the different states after p layers for the initial state described by equation (4.3). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



(a) Probabilities for the states after p=9 layers.

(b) Probabilities for the states after p=10 layers.

Figure 4.13: Probabilities for the different states after p layers for the initial state described by equation (4.3). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.

All these distributions results in different expectation values of the energy. How this expectation value changed with each layer can be seen in figure 4.14.

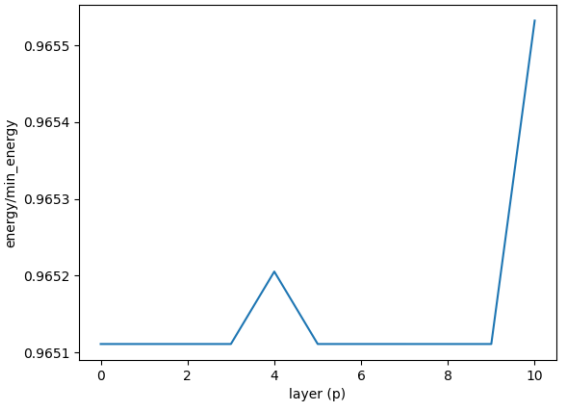


Figure 4.14: Evolution of the expectation value of energy calculated as in equation 2.8 for each layer, compared to the energy of the best state when starting in a superposition of all good states.

The small difference in energy between all the good states makes the state staying the same for almost every layer. However, after $p = 4$ layers the expectation value of the energy has increased a little bit before returning to the same value again. The difference is very small at 0.001. The expectation value increases a little bit more after $p = 10$ layers. This can be seen in figure 4.13b where the two states with the highest probability are the two best states. The superposition of all the good states is not the optimal initial state for the SWAP mixer. The total energy is already

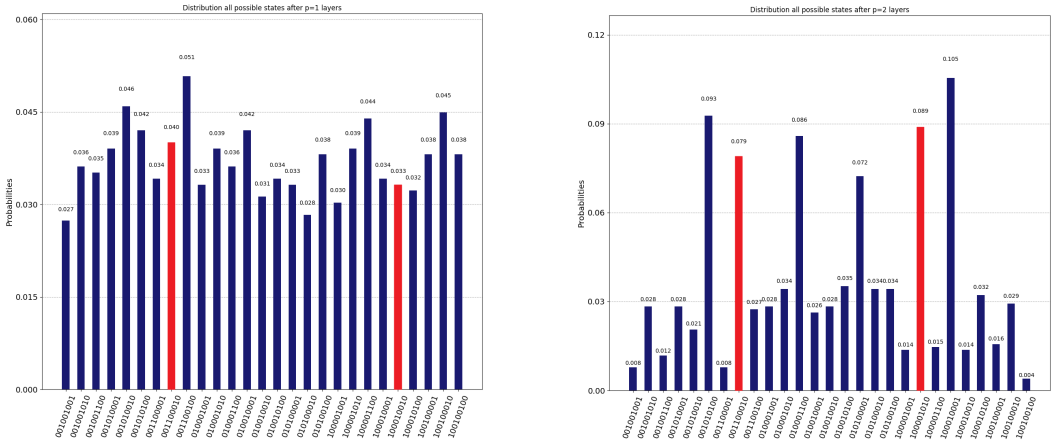
almost the best making it hard to improve. A last interesting test would be to see how a superposition of all the possible states performs.

4.1.5 Starting in a superposition of all the states in subspace

In the last test the performance of the SWAP mixer when starting in a superposition of all the possible states are examined. There is a total of 27 states which makes it the optimal test for the SWAP mixer. With these states the initial state can be described as in equation (4.4).

$$\begin{aligned} s_0 = &\frac{1}{\sqrt{27}}(|001001001> + |001001010> + |001001100> + |001010001> + |001010010> + \\ &|001010100> + |001100001> + |001100010> + |001100100> + |010001001> + \\ &|010001010> + |010001100> + |010010001> + |010010010> + |010010100> + \\ &|010100001> + |010100010> + |010100100> + |100001001> + |100001010> + \\ &|100001100> + |100010001> + |100010010> + |100010100> + |100100001> + \\ &|100100010> + |100100100>) \end{aligned} \tag{4.4}$$

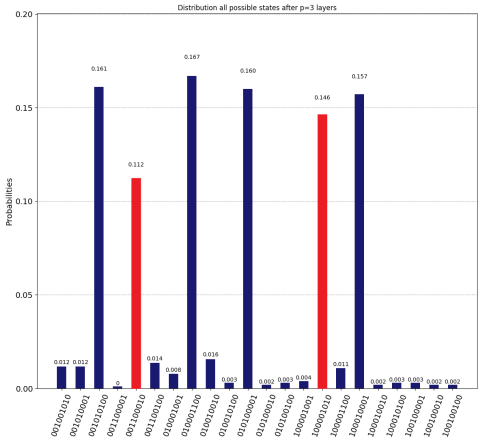
The following figures describes how the distribution of the states change with each layer p.



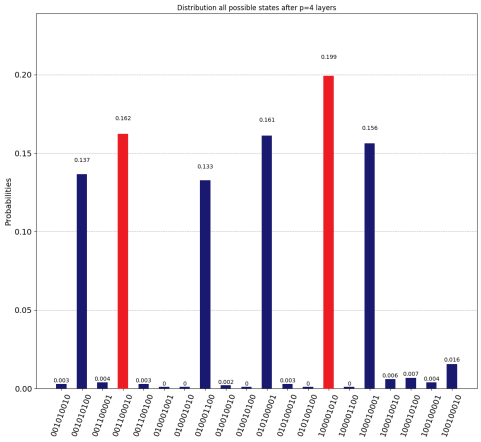
(a) Probabilities for the states after p=1 layers.

(b) Probabilities for the states after p=2 layers.

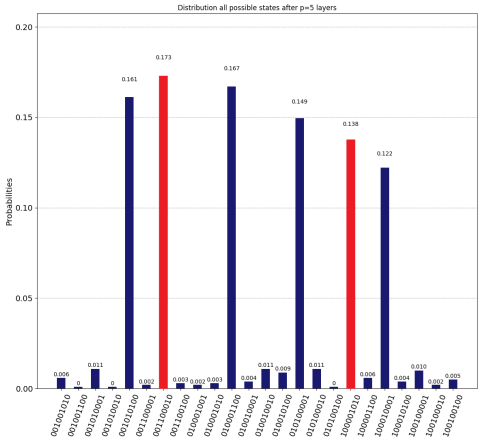
Figure 4.15: Probabilities for the states after p layers for the initial state consisting of all the possible states as described in equation (4.4). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



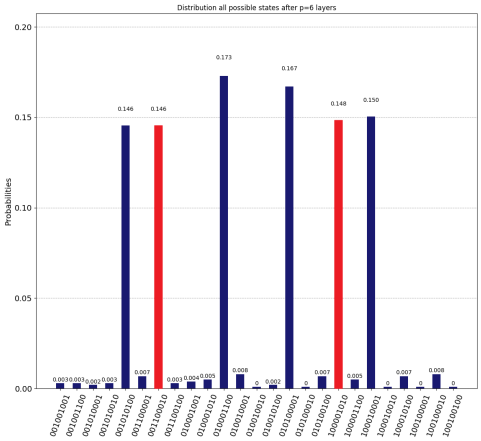
(a) Probabilities for the states after p=3 layers.



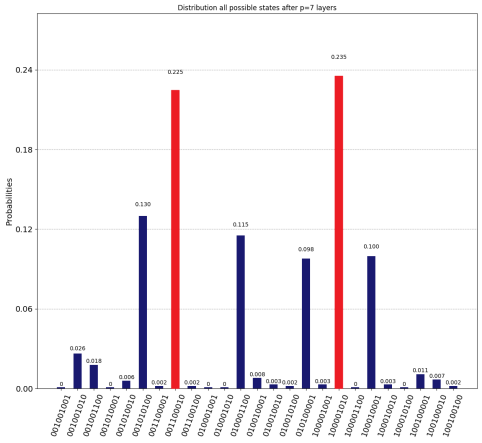
(b) Probabilities for the states after p=4 layers.



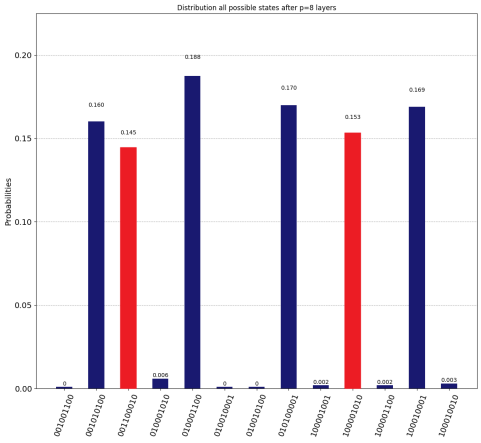
(c) Probabilities for the states after p=5 layers.



(d) Probabilities for the states after p=6 layers.

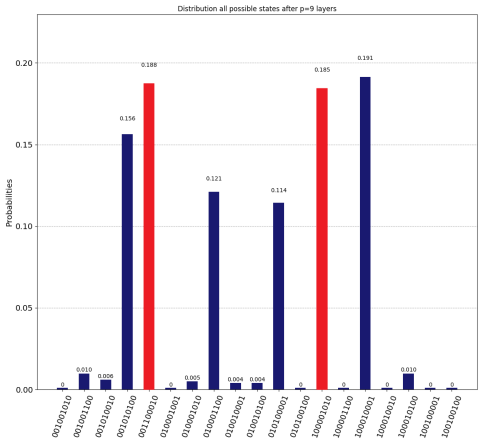


(e) Probabilities for the states after p=7 layers.

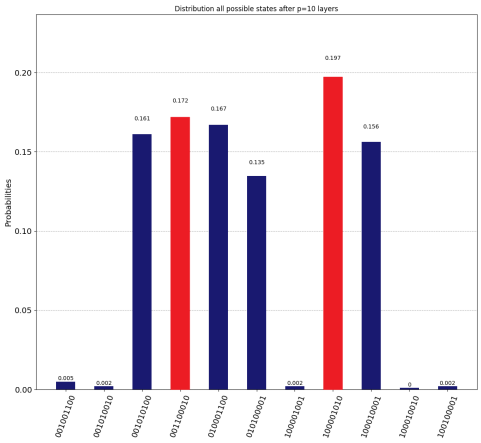


(f) Probabilities for the states after p=8 layers.

Figure 4.16: Probabilities for the states after p layers for the initial state consisting of all the possible states as described in equation (4.4). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.



(a) Probabilities for the states after p=9 layers.



(b) Probabilities for the states after p=10 layers.

Figure 4.17: Probabilities for the states after p layers for the initial state consisting of all the possible states as described in equation (4.4). Above each bar the probability can be seen. The red bars are the states with the lowest energy as described earlier.

All these distributions results in different expectation values of the energy. How the expectation value changed with each layer can be seen in figure 4.18.

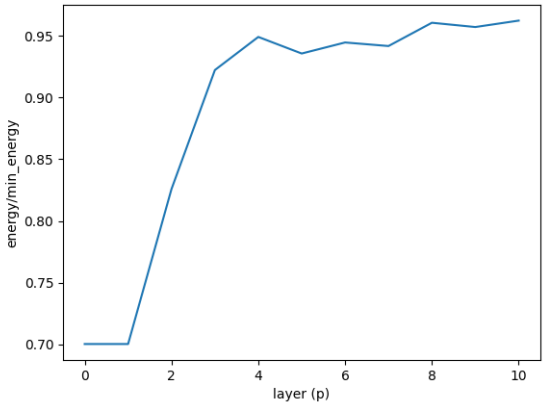


Figure 4.18: Evolution of the expectation value of energy calculated as in equation 2.8 for each layer, compared to the energy of the best state when starting in a superposition of all states in subspace.

With the initial state of all the possible states, the SWAP mixer has found the good states after $p = 2$ layers in figure 4.15b. However, after that, the performance becomes very similar to the earlier test for all the good states. With the probability of all the good states staying at 1/6 through all the layers of operations. This is quite strange, we expect that more start states should make it more likely to end up in a better state. The reason for this can be that in the case with only bad states

the energy difference is pretty significant and makes it easier to SWAP to the best good state. When there is already a probability for the good states that are a little worse the energy difference for the total system before and after a SWAP is much lower and hence preventing a SWAP.

4.1.6 Comparison of all the initial states

How the performance of the SWAP mixer depend on the initial state can be seen in figure 4.19. This figure describes how the expectation value of the energy changes with layer for all the initial states.

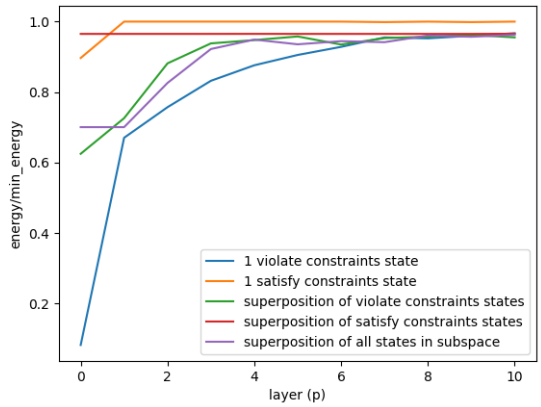


Figure 4.19: Comparison of how the expectation value of the energy changes with layers p for every different initial states.

The performance of the single good state achieved the best energy compared to the others. But as stated earlier it does not utilize the advantages of a quantum computer. Instead, the best initial state were the ones starting in bad states. With these initial states, the probability of the best states was significantly bigger than the other initial states for all the good states and all possible states.
For $p = 9$ and $p = 10$ in figure 4.3e and figure 4.3f the two best states end up with a higher probability than the rest of the good states. This can also be seen for other initial states such as in figure 4.9a and figure 4.9b. This means that QAOA with a SWAP mixer is quite good at sorting out the bad states and find the best states. However, it cannot swap to end up in the best state with 100% probability. This is probably because the energy difference between the good states is quite low. This can be seen in figure 4.4 and figure 4.10 where the energy of the states increases to almost the optimal and then staying the same. The SWAP mixer seems to have a little problem when the energy difference is low. The reason for this is probably that when the energy is almost the lowest possible a SWAP may not necessarily lower the energy. A SWAP that is applied to a state with a little higher energy to get a better state with a lower energy is also applied to all the other states and can increase their energy.

5
Conclusion

In this paper, I have tested using a SWAP mixer for QAOA on an instance of the Traveling salesman problem. With the SWAP mixer a subspace of the whole state space could be generated and the SWAP mixer could preserve this subspace. With this subspace less solutions had to be explored compared to if the ordinary bit flip operator in the ordinary QAOA, that explores the whole space would have been used. Instead of the $2^9 = 512$ states that have to be explored in the ordinary QAOA, the SWAP mixer only had to explore $3^3 = 27$ states. This difference in number of states are something that would increase with number of qubits making the SWAP even more suited for particular problems that can utilize such a mixer.
Overall the SWAP mixer performed pretty well and could distinguish the good states from the bad states that broke the constraints. To fully test the performance of the SWAP mixer and see if it can find the best state the difference in energies between the states has to increase. One way to do this is to increase the number of cities visited and with that increase the number of qubits used. A first test would be to test for 5 cities and 16 qubits. However, an increase in qubits implies an increase in the number of required operations to simulate a quantum circuit representing the new algorithm. This increase of quantum operations increased the operational time of the program significantly taking more than 5 days instead of the 1 hour for the 9 qubit version and required a considerable amount more memory. A way to solve this problem would be to use a different kind of simulator that is more efficient in simulating a quantum computer. A different approach would be to have a list of pre-defined parameters γ_p and β_p to limit the optimizer part of the algorithm. It is pretty hard to see if these pre-defined parameters are good parameters or bad compared to all possible. Another way to speed up the algorithm would be to use machine learning to predict the parameters. [15] A quantum circuit is expensive to simulate on a classical computer which is of course why we would like to use a real quantum computer.
The performance of the SWAP mixer showed potential and should be a contender for mixing in more complex optimization problems than the TSP. An example is the capacitated vehicle routing problem (CVRP) that is based on the traveling salesman problem but instead of one person, you have multiple vehicles that together want to deliver an amount of load to different stops. The goal is to find the optimal route for every vehicle given a max freight load for each vehicle and that every stop is visited once. [16] [17]
When using electronic equipment for measuring quantities noise is a problem often occurring. An interesting thing to look at would be to see how noise affects the algorithm and how you can implement the algorithm to limit the impact of

the noise. This has earlier been tested on the Max-κ-Colorable-Subgraph problem for the Quantum Alternating Operator Ansatz [18]. Qiskit has a backend called QasmSimulator that implements noise to the quantum circuit [12]. However, adding noise to the computation may result in a state outside the subspace generated by the SWAP mixer. This will probably make the SWAP mixer useless because it is defined to work on states in that particular subspace.

Bibliography

[1] Michael R. Swaine, Paul A. Freiberger, David Hemmendinger, and William Morton Pottenger. *computer*. https://www.britannica.com/technology/computer. Accessed: 2021-06-01. 2021.

[2] Ingela Roos. "Now the quantum computer will become reality". In: *Chalmers magasin* 1 (2018), pp. 16–20.

[3] Miriam Haart and Charlie Hoffs. "Quantum Computing: What it is, how we got here, and who's working on it." In: (Mar. 2019).

[4] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. *A Quantum Approximate Optimization Algorithm*. 2014. arXiv: 1411.4028 [quant-ph].

[5] Zhihui Wang, Nicholas C. Rubin, Jason M. Dominy, and Eleanor G. Rieffel. "XY mixers: Analytical and numerical results for the quantum alternating operator ansatz". In: *Physical Review A* 101.1 (Jan. 2020). ISSN: 2469-9934. DOI: 10.1103/physreva.101.012320. URL: http://dx.doi.org/10.1103/PhysRevA.101.012320.

[6] QuTech Quantum Inspire. *Swap Gate*. https://www.quantum-inspire.com/kbase/swap/. Accessed: 2021-06-01.

[7] Stuart Hadfield, Zhihui Wang, Bryan O'Gorman, Eleanor Rieffel, Davide Venturelli, and Rupak Biswas. "From the Quantum Approximate Optimization Algorithm to a Quantum Alternating Operator Ansatz". In: *Algorithms* 12.2 (Feb. 2019), p. 34. ISSN: 1999-4893. DOI: 10.3390/a12020034. URL: http://dx.doi.org/10.3390/a12020034.

[8] Routific. *Understanding The Travelling Salesman Problem (TSP)*. https://blog.routific.com/travelling-salesman-problem. Accessed: 2021-06-01. Jan. 2020.

[9] Shabnam Sangwan. "Literature Review on Travelling Salesman Problem". In: *International Journal of Research* 5 (June 2018), p. 1152.

[10] Pontus Vikstål. "Application of the quantum approximate optimization algorithm to combinatorial optimization problems". PhD thesis. Gothenburg, Sweden: Chalmers University of Technology, 2020.

[11] Qiskit Development Team. *Learn Quantum Computation using Qiskit*. https://qiskit.org/textbook/preface.html. Accessed: 2021-06-01.

[12] Qiskit Development Team. *Simulator*. https://qiskit.org/documentation/tutorials/simulators/1_aer_provider.html. Accessed: 2021-06-01.

[13] SciPy.org. *scipy.optimize.differential_evolution*. https://docs.scipy.org/doc/scipy/reference/reference/generated/scipy.optimize.differential_evolution.html#scipy.optimize.differential_evolution. Accessed: 2021-06-01.

[14] Thomas Bergamaschi. *Quantum Approximate Optimization Algorithms on the 'Traveling Salesman Problem'*. https://medium.com/mit-6-s089-intro-to-quantum-computing/quantum-approximate-optimization-algorithms-on-the-traveling-salesman-problem-703b8aee6624. Accessed: 2021-06-01.

[15] Mahabubul Alam, Abdullah Ash-Saki, and Swaroop Ghosh. *Accelerating Quantum Approximate Optimization Algorithm using Machine Learning*. 2020. arXiv: 2002.01089 [cs.ET].

[16] Zuzana Borcinova. "Two models of the capacitated vehicle routing problem". In: *Croatian Operational Research Review* 8 (Dec. 2017), pp. 463–469. DOI: 10.17535/crorr.2017.0029.

[17] David Fitzek. et al. "A quantum approximate optimization formulation for the heterogenous vehicle routing problem". 2021.

[18] Michael Streif, Martin Leib, Filip Wudarski, Eleanor Rieffel, and Zhihui Wang. "Quantum algorithms with local particle-number conservation: Noise effects and error correction". In: *Physical Review A* 103.4 (Apr. 2021). ISSN: 2469-9934. DOI: 10.1103/physreva.103.042412. URL: http://dx.doi.org/10.1103/PhysRevA.103.042412.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY