UNIVERSITÄT SIEGEN

Naturwissenschaftlich Technische Fakultät

Correlations and Dynamics of Spin Systems in Quantum Simulations

Masterarbeit
zur Erlangung des akademischen Grades
Master of Science

Jens Borgemeister
1166390

Universität Siegen
Department Physik

Gutachter:
Priv.-Doz. Dr. Kleinmann
Prof. Dr. Otfried Gühne

Betreuer:
Dr. H. Chau Nguyen

Abgabedatum: 02. August 2022

Contents

1 Introduction 1

2 Current Quantum computers 2
2.1 Gate based quantum computers . . . 2
2.2 Adiabatic quantum computers . . . 4
2.3 Short outlook . . . 4

3 Introduction to the IBM quantum hardware 6
3.1 Cloud based quantum computing . . . 6
3.2 Quantum Volume . . . 7
3.3 Circuit Layer Operations Per Second . . . 8

4 Introduction to Qiskit 9
4.1 Multi-qubit basis ordering convention . . . 9
4.2 Circuit for preparing a Bell state . . . 10
4.3 Basics of quantum simulation in Qiskit . . . 12
4.4 Measurements and state tomography . . . 14
4.5 Running a quantum circuit on the IBM hardware . . . 16
4.6 Qiskit Transpiler . . . 18
4.7 Readout error correction . . . 25
4.8 Advanced gates . . . 28

5 Case study: Preparation of mixed Werner state and Bell diagonal states 30
5.1 Simulating a Werner state using classical post-processing . . . 30
5.2 Preparing a Werner state using Bell Diagonal states . . . 32

6 Simulation of unitary dynamics 34
6.1 Exactly solvable system - Transverse Ising Hamiltonian . . . 34
6.2 Simulation of the Ising model on a quantum computer . . . 39
6.3 Trotter method . . . 47

1 Introduction

In the recent years, we have seen much progress in quantum computing. In 2019 Google made headlines with showing quantum supremacy for the first time. Since then it has been vigorously debated how quantum computers would once affect our daily lives. In this thesis we provide an overview of one of the most important architectures of quantum computers currently under development at IBM.

We start with an overview of current available or proposed architectures. Then we have a closer look at some of IBM's quantum computers which are available to us in the cloud. To be able to get an impression of what we can do with them, we will look in detail at what is behind the different parameters and benchmarks by which they, and also the devices of some other quantum computing companies, are characterized.

In the third section we explain how to work with Qiskit. This is a python library introduced by IBM, but now also adapted by a few other quantum computing manufacturers as well. Qiskit allows us to create and run quantum circuits, which are the instructions that we use to prepare, manipulate and measure quantum states on quantum computers. Within this section, we cannot explain all details of Qiskit; but it should be enough to start working with Qiskit on also get an understanding what the circuit compiler, also called Transpiler, does.

Finally we will look at some physical examples. First we look at how to prepare a Werner state, which is a state combined of the totally mixed and a maximally entangled Bell state. Then we will look in detail at how to simulate a Ising spin chain on a quantum computer, and how to prepare its ground and thermal states for different parameters.

2 Current Quantum computers

2.1 Gate based quantum computers

A universal gate-based quantum computer should be able to start in a given quantum state and then carry out any unitary transformation on it. DiVincenzo gave five necessary criteria for the realization of a universal quantum computer [1]:

A scalable physical system with well characterized qubits. Scalability is a very difficult to achieve, as one wants to increase the number of qubits while maintaining a good connectivity between the qubits[1] and achieving low error rates. Also the measurement procedure has to be considered. Which physical system has the best scalability is the focus of current research, the best current systems reach hundreds of physical qubits [2].

The ability to initialize the state of the qubits to a simple fiducial state, such as $|000...\rangle$ So there has to be a physical process which drives the system into a known state. This might be cooling the system into the ground state, using lasers to excite to specific states or if the state after the measurement is known just doing a measurement.

Long relevant decoherence times, much longer than the gate operation time. The decoherence of the qubit in reality can be considered as consisting of two main processes [3], the first being the decay of the excited state $|1\rangle$ into the ground state $|0\rangle$, which is described by the energy relaxation time T_1. The second can be thought of as a random rotation of the state along the equator of the Bloch sphere, so that the state $|+\rangle = \frac{1}{2}(|0\rangle + |1\rangle)$ evolves into a mixture of $|0\rangle\langle 0|$ and $|1\rangle\langle 1|$. How fast this process takes place is described by the coherence time T_2, which can be thought of as the decay constant for the expectation value of the X basis measurement (with state $|x\rangle$ initially prepared) [4].

A "universal" set of quantum gates. Having the CNOT gate (or other two-qubit gate(s) form which one can create the CNOT by using any single qubit gates) and a set of single qubit gates, from which any single qubit gate can be created, is enough for the universality [5]. For example, for the IBM quantum machines [6] the basis gates are the CNOT and the single qubit gates

$$\mathrm{R_Z}(\lambda) = \begin{pmatrix} e^{-i\frac{\lambda}{2}} & 0 \\ 0 & e^{i\frac{\lambda}{2}} \end{pmatrix}, \sqrt{\mathrm{X}} = \mathrm{SX} = \frac{1}{2}\begin{pmatrix} 1+i & 1-i \\ 1-i & 1+i \end{pmatrix} \text{ and } \mathrm{X} = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}. \quad (1)$$

A gate is realized by applying a Hamiltonian for a given time, so the choice of basis gates depends on which Hamiltonians can be turned on for the chosen underlying system.

[1] Which means in order to apply a two qubit operation on any two qubits in the system we only have to swap a low number of qubits

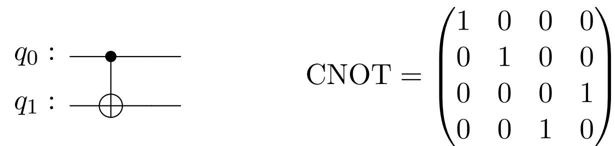

Figure 1: Circuit symbol and matrix for the CNOT gate.

A qubit-specific measurement capability. This means that there has to be a measurement process which can measure individual qubits (usually this is in Z basis and one applies a rotation before the measurement if any other basis is required).

Let us now have a short look at some of the different ways to implement a universal quantum computer:

Trapped Ions

One is by trapping ions using an electromagnetic field (Paul trap) [7]. To initialize the ions into a specific state, a laser excites the ions of all except one state into a state of higher energy, through the decay process eventually all ions find their way to the state not addressed by the laser. This state then corresponds to $|0\rangle$ and another state is chosen to correspond to $|1\rangle$. The single qubit gates are also implemented by coupling the qubit to appropriate external lasers. The swap gate can be implemented by physically moving the ions. Instead of the CNOT here the so-called Mølmer-Sørensen gate is used, as this has the highest two-qubit gate fidelities for this system and also forms a universal gate set together with single qubit gates. This gate is implemented by applying one laser on all qubits simultaneously as well as a laser for each qubit specifically. The measurement is done by taking a photo with a CCD camera.

Nuclear spin qubit

Also for the nuclear spin/neutral atom qubit [8][9] lasers are used to control the atoms. As the atoms have neutral charge, the trapping occurs by light with a grid of lasers. For the qubits then the nuclear spins of the trapped atoms are used. The single qubit gates are implemented by shifting the atoms resonance frequency with lasers and then applying a microwave field. The two-qubit gate is the controlled Z gate (C_Z). To implement this the two neighboured atoms are excited to Rydberg states, as otherwise the interaction between them would be very low. Readout is also done by a camera. An advantage of this method is that even millions of nuclear spins could fit in a very small space.

Semiconductor Spins

Another way of realizing a qubit is by using the spin of electrons in semiconductors [10]. The electrons are confined in narrow potential minima, where the energy level of the minimum as well as the barriers can be controlled by electrodes. For the measurement the potential is modified so that the electrons can tunnel to other sites if they are in the

spin up state. This is also used for initialization, as after the measurement the state of all the electrons is known. The gates are implemented by magnetic or electric excitation (mostly for single qubit gates) or by modifying the gate voltages so that neighboring spins can interact (for two qubit gates).

Advantages are that the production of the quantum processors may take advantage of existing production technology for classical ones, as similar silicon wafers have been intensively used in silicon industry [11]. Also the chips are smaller and can operate at higher temperatures than superconducting charge qubits (see below). The energy relaxation time is relatively long ($T_1 > 1\,\mathrm{s}$), while the coherence time T_2 is in the same order as for the superconducting qubits. [2]

Superconducting qubits

The Josephson junction consists of two superconductors, which have a thin insulator between them. This leads to a tunneling current, which presents even if no external voltage is applied.

For the **flux qubit** one is interested in exactly this case, as without an external voltage the current can be flowing in both directions, even at the same time. So we have two states with superposition possible, which gives us a well characterized qubit. Initialization is done by cooling the system into the ground state. Gate operations are implemented by applying microwave pulses and the readout is implemented by measuring the phase or amplitude of a microwave driving field [12].

The **transmon qubit** [13] is using two parallel Josephson junctions among capacitors and inductors. This allows the qubit to be operated with almost an insensitivity to charge noise (fluctuating electric field at the qubit), which greatly increases the decoherence times of the qubit. The transmon qubit is used in the IBM and Google quantum chips.

2.2 Adiabatic quantum computers

Adiabatic quantum computers work by preparing an initial Hamiltonian with the system in its known ground state and then changing the Hamiltonian into the desired Hamiltonian. If the change is slow enough and the energy gap to the lowest energy excited state is not closed, the Hamiltonian will stay in the ground state. Then one has prepared the ground state of the new Hamiltonian. This can be used to solve difficult optimization problems [14] or for machine learning sampling [15].

Quantum annealing is a simplified implementation of this, excepting that due to experimental limitations one cannot prepare the exact ground state, but just an ensemble of low energy states [16]. This type of quantum processors is pioneered by D-Wave [17].

2.3 Short outlook

It is difficult to make predictions when a certain number of qubits can be reached or when quantum computers would achieve supremacy for a specific task. According to the goals

[2] For the latest chip by Intel (2022) the coherence time T_2 is over $3\,\mathrm{ms}$

of IBM and other companies, we should see a million qubit machine before 2030 [18]. But it remains to be seen if this plan can actually be realized. If it can, quantum computers might become useful for solving many real world problems (depending on connectivity and noise in the device).

A big advancement towards commercial usability would also be to implement error correction codes, which combine a set of physical qubits to logical qubits and would result in a much less noisy system. But for this a good connectivity between qubits and a large qubit overhead for error correction would be required.

The current systems (or quantum simulators on classical computers[3]) can be used to test applications which can be later scaled up to thousands or millions of (error-corrected) qubits. Of course one has to keep in mind that any classical (post)processing should be scalable as well.

[3]These are getting to their limits in the region of $30 - 100$ qubits depending on the hardware and required simulation precision.

3 Introduction to the IBM quantum hardware

3.1 Cloud based quantum computing

IBM has 22 noisy intermediate scale quantum (NISQ) computers with from 5 to 127 qubits and 5 cloud based quantum simulators.[4] The NISQ computers are based on the superconducting transmon architecture. The 5 qubit and some of the 7 qubit devices are freely available, more devices can be accessed by a premium plan.[5]

At the info page about the current systems,[6] each is listed with its system status, processor type, number of qubits, QV (Quantum Volume) and CLOPS (Circuit Layer Operations Per Second), the last two being benchmarks that will be explained later.

By clicking on one of the devices one can see the qubit map, which shows which qubits are connected, as well as the latest calibration data. This includes for each qubit the coherence times T_1 and T_2, information about frequency (split into the frequency ω, describing the energy difference between the two levels of the qubit, and the aharmonicity δ, which describes the transition frequency for higher energy levels of the transmon according to $\omega_{j+1} - \omega_j = \omega + \delta j$) [19], readout errors (state-independent average, as well as in detail for measuring $|0\rangle$ if $|1\rangle$ was prepared and vice versa), readout time and single qubit gate errors. Also for each connection the CNOT error and CNOT gate time is available. For each of these values there is an average over all qubits/connections given, too.

In addition to the information regarding each qubit and connection it also shows how many pending jobs the device has and below the limitations for the jobs one wants to send (more on this later).

The gate times for single qubit gates can only be looked up using Qiskit. But usually the single qubit execution times are negligible, as for the SX and X gate they are around 1/10 of the CNOT gate execution times. The RZ gate is implemented virtually, and therefore does not take any time.

In the following table, the relevant execution times for *ibm_nairobi* are presented (see eq. 1 for matrix form of these gates):

.	RZ	SX, X	CNOT	Readout	T1	T2
Time (avg.)	0 s	35.6 ns	307 ns	5.35 µs	162.67 µs	77.78 µs

Table 1: Execution times for *ibm_nairobi* as of June 29th 2022. The single qubit gate times are equal for all qubits, for the other times the average over all qubits is given.

[4] as of June 2022
[5] https://www.ibm.com/quantum/access-plans
[6] https://quantum-computing.ibm.com/services?services=systems

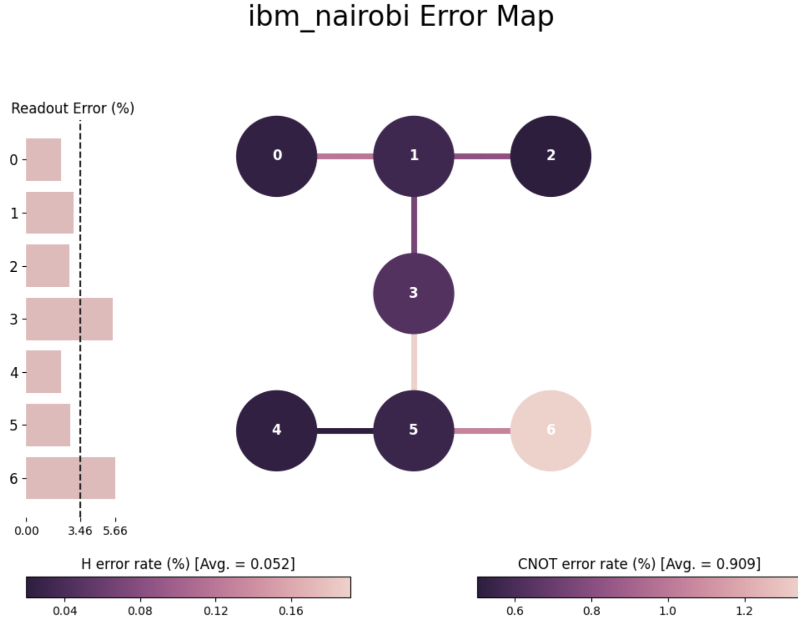

Figure 2: Error map for the 7 qubit IBM device *ibm_ nairobi*, showing readout and gate errors

3.2 Quantum Volume

IBM has introduced the Quantum Volume (QV) as a benchmark for to reliability of NISQ devices [20]. It shows how well a random circuit of equal qubit count and depth (layers of two-qubit gates[7] needed for this circuit on an ideal quantum computer, so if each qubit is connected to any other) runs on this device.

To create a d qubits and d layers square random circuit C, for each of the d circuit layers one applies a random permutation of the qubit indices and then adds random two-qubit gates[8] for every 2 consecutive qubits in the permutation (for odd d, the last qubit is left idle). This corresponds to connecting each qubit to exactly another qubit by a random two-qubit gate, repeated for each layer.

Then we simulate the circuit C on a classical computer to get the output probabilities $p(x)$ for each possible measurement result $x \in \{0,1\}^d$. We are interested in the outputs x corresponding to the larger half of probabilities (so all which are larger than the median), called the heavy outputs

$$H_C = \{x \in \{0,1\}^d | p(x) > p_{median}\} . \tag{2}$$

[7]Here arbitrary two-qubit gates are considered, not just the basis gates of a quantum computer like one usually does when given the depth of a circuit **write this or already clear enough?**

[8]These random two-qubit gates are sampled from the Haar measure of SU(4).

Then the probability that the measurement outcome is a heavy output is

$$\mathbb{P}_C(x \in H_C) = \sum_{p(x) > p_{median}} p(x) \tag{3}$$

where $\mathbb{P}_C$ is the probability measure for the measurement outcomes of circuit C. One can show that for arbitrary random circuits C on an ideal device [21], it holds that

$$\forall \epsilon > 0 : \; \mathbb{P}_d \left(\mathbb{P}_C(x \in H_C) < \frac{1 + \ln 2}{2} - \epsilon \right) < \exp(-O(d)) \tag{4}$$

where $\mathbb{P}_d$ is the probability measure over all d square random circuits. This equation expresses that the probability for a random d-square circuit not fulfilling $\mathbb{P}_C(x \in H_C) \geq (1 + \ln 2)/2 \approx 0.85$ is decreasing exponentially in d.

This is then used to verify if the quantum device can run the circuit well enough. One requires that - with a confidence level of 2σ - the probability of a result from the experiment being a heavy output from the simulation, is larger than $2/3$. This value is chosen by convention, in order to have a limit which is far enough from 0.5, which would be the probability for sampling a heavy outcome on a very noisy device which ends in the totally mixed state.

The largest number of qubits d for which this test passes then defines the quantum volume of the device

$$\mathrm{QV} = 2^d \,. \tag{5}$$

3.3 Circuit Layer Operations Per Second

The Circuit Layer Operations Per Second (CLOPS) is a benchmark for the speed of NISQ devices [22].

100 random square circuits with $d = \ln QV$ qubits and layers, so the largest which run reliable on the device, are generated to measure the CLOPS. The only difference to the circuits used for determining the quantum volume is that the random two-qubit gates are not immediately randomly sampled, but kept as arbitrary two qubit gates. This is achieved by implementing them as parameterized gates.[9] Each circuit is then run 100 times for 10 different random parameter sets.

The total number of circuit layers is then divided by the time t it takes from the quantum device receiving the circuits (compiled into circuits consisting of its basis gates) until receiving the results:

$$\mathrm{CLOPS} = \frac{\ln QV \cdot 100 \cdot 10 \cdot 100}{t} \tag{6}$$

[9] These are gates with parameter sets. Values are assigned to the parameters after compiling the circuit, so the circuit is only compiled once for all parameters. This speeds up the compiling, but as this time is not counted here, there should not be any time advantage over using totally different circuits.

4 Introduction to Qiskit

The Python library Qiskit [23][24] can be used to execute a quantum ciruit on the IBM quantum hardware[10] or to run a simulation, which can be done locally or in the cloud. This section will be an introduction how to use Qiskit. In the beginning of this section we will use Qiskit to create a quantum circuit which prepares the Bell state $|\Psi^-\rangle$ and then use this example to give a short introduction into Qiskit.

To use Qiskit you can either choose the jupyter notebook based online version offered from IBM[11] or install the Qiskit python library in your python environment. The first has the advantage of having already installed all required packages, while the second will probably be more convenient for larger programs.

4.1 Multi-qubit basis ordering convention

We first have a look at the ordering convention for the basis states in Qiskit, which is different from the one usually used.

The states $|0\rangle$ and $|1\rangle$ can be represented by the matrices

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \text{ and } |1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{7}$$

When we have multi-qubit states, they are expressed as tenorproducts of single qubit states. Usually this tensor product we would have the form

$$|a_0\rangle_0 \otimes |a_1\rangle_1 \otimes \ldots |a_n\rangle_n \equiv |a_0 a_1 \ldots a_n\rangle \ , \tag{8}$$

so $|01\rangle$ would be the first qubit in $|0\rangle$ and the second in $|1\rangle$.

In Qiskit this order is reversed, so we have

$$|a_n\rangle_n \otimes |a_{n-1}\rangle_{n-1} \otimes \ldots |a_0\rangle_0 \equiv |a_n a_{n-1} \ldots a_0\rangle \ , \tag{9}$$

which means $|01\rangle$ would be the first qubit in $|1\rangle$ and the second in $|0\rangle$.

When ordering those states, one looks at the integers which they would represent if the labels were binaries, so the position in a list of all basis states is given by

$$\sum_{k=0}^{n} a_k 2^k \ . \tag{10}$$

This affects the unitaries for multi-qubit gates as well as for complete circuits and changes the ordering of measurement results.

[10] Some other providers also support Qiskit.

[11] https://lab.quantum-computing.ibm.com/

4.2 Circuit for preparing a Bell state

We want to prepare the Bell state, which is given by

$$\left|\Psi^{-}\right\rangle = \frac{1}{\sqrt{2}}\left(|01\rangle - |10\rangle\right). \tag{11}$$

In Qiskit's state labeling convention (see section 4.1) this is

$$\left|\Psi^{-}\right\rangle = \frac{1}{\sqrt{2}}\left(|10\rangle - |01\rangle\right). \tag{12}$$

State preparation using a quantum circuit

This can be done by the following circuit:

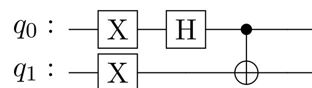

Figure 3: Circuit for preparing $|\Psi^{-}\rangle$

The Pauli X gates put both qubits into state $|1\rangle$, then the Hadmard gate H transforms the first qubit into the state $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. So the CNOT - which switches the state of the second qubit if the first is in $|1\rangle$ - finally gives the state $|\Psi^{-}\rangle$.

Now we create this circuit in Qiskit

```
from qiskit import QuantumCircuit

circ = QuantumCircuit(2)
circ.x([0, 1])
circ.h(0)
circ.cx(0, 1)
```

In the third line the quantum circuit is initialized with 2 qubits and without classical bits, as we do not have to store any measurement result yet. The general form of this function is

```
QuantumCircuit(n_qubits, n_classical_bits, name='circuit name')
```

In lines 3 to 6 the quantum gates are implemented by calling methods of the *QuantumCircuit* instance *circ*. The methods for all predefined gates can be found in the Qiskit documentation.[12] The number of arguments depends on if it is a single- or two-qubit gate and if the gate needs additional parameters (e.g. a phase gate). The qubit indices can be given as integers (like in lines 5 and 6), or as lists of qubit indices, if one wants to apply the same gate multiple times (like in line 4, where the X gate is implemented on both qubits).

[12]https://qiskit.org/documentation/tutorials/circuits/3_summary_of_quantum_operations.html

We verify that the desired circuit was generated by calling the draw method. There are different drawing options in Qiskit:

```
circ.draw(output='text')  # print circuit as text, default option
circ.draw(output='mpl').show()  # generate image using matplotlib library
circ.draw(output='latex_source')  # give LaTeX source code
```

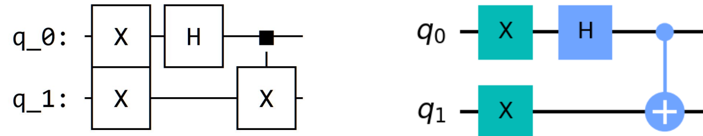

Figure 4: Circuit drawn using the `output='text'` (left) and `output='mpl'` (right) option. For a circuit drawn with the `output='latex_source'` option see Figure 3.

State preparation using the Initialize method

Another way to prepare a state is to use Qiskit's initialization method. This uses a recursive initialization algorithm based on [25] to generate a circuit which prepares the desired state.

First we express the state by giving the coefficients of the Z-basis states. Here we have to consider section 4.1 for the correct state labelling and ordering. For the state $|\Psi^-\rangle$ this is

$$\left|\Psi^-\right\rangle = \frac{1}{\sqrt{2}}\left(|10\rangle - |01\rangle\right) \Rightarrow \begin{pmatrix} 0 & \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \end{pmatrix}^T \tag{13}$$

The initialization method is then used to create a circuit which prepares this state. In addition to the desired state vector it is given a list containing the qubit indices of the qubits on which the state should be prepared.

```
from qiskit import QuantumCircuit, execute, Aer
import numpy as np

desired_vector = [
    0,
    -1 / np.sqrt(2),
    1 / np.sqrt(2),
    0]

circ = QuantumCircuit(2)
circ.initialize(desired_vector, [0, 1])
```

Now we look at the quantum circuit that was generated (Figure 5). We only see one custom gate, which has the individual gates 'hidden' within. To see the detailed circuit like in Figure 6 we call

```
circ.decompose()
```

iteratively for each layer of custom gates.

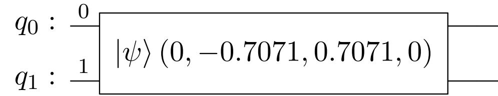

Figure 5: Circuit for preparing $|\Psi^-\rangle$, generated by *initialize* method.

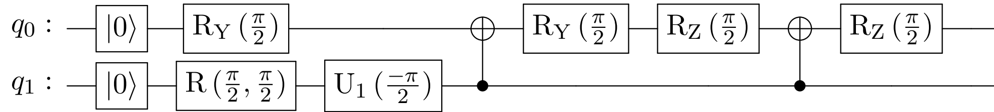

Figure 6: Circuit for preparing $|\Psi^-\rangle$, generated by *initialize* method. Decomposed into standard gates. The $|0\rangle$ gate is a reset gate, which resets the qubit into the $|0\rangle$ state.

Comparing the circuit with the circuit from Figure 3 we see that it uses two CNOT gates instead of one. So it did not generate the shortest possible circuit, but this is expected from an algorithmic approach. For higher qubit numbers the number of CNOTs in the generated circuit stays at about twice the optimal number needed to prepare the state [25].

In Qiskit's *aer_simulator* and *statevector_simulator* the initialize operation is implemented directly, so no unrolling into single- and two-qubit gates is needed.

4.3 Basics of quantum simulation in Qiskit

Qiskit Aer gives the tools for the simulation of quantum circuits. The *statevector_simulator* gives out the quantum state prepared by the circuit, the *unitary_simulator* calculates the unitary of the circuit and the *aer_simulator* simulates the circuit being run on a quantum computer.

Statevector simulator

To use the statevector simulator we add the following lines to the example from the previous section 4.2, where we created the quantum circuit *circ* (which prepares the state $|\Psi^-\rangle$):

```
1 from qiskit import Aer
2
3 backend = Aer.get_backend('statevector_simulator')
4 job = backend.run(circ)
5 result = job.result()
6 outputstate = result.get_statevector(circ, decimals=3)
7 print(outputstate)
```

First Qiskit Aer is imported and the *statevector_simulator* is selected as backend. Then the simulation is run and the statevector retrieved. Like expected, this returns

```
Statevector([ 0.   +0.j, -0.707+0.j,  0.707-0.j,  0.   +0.j], dims=(2, 2))
```

which corresponds to the state $|\Psi^-\rangle$. Here $.j$ is the imaginary unit.

The statevector simulator only works for unitary circuits, so for circuits without measurement or qubit reset instruction.

Unitary simulator

With the *unitary_simulator* one can calculate the unitary matrix of a quantum circuit. This is particularly useful when we implement a multi-qubit unitary as a quantum circuit and want to verify it. Obviously this simulator only works if the circuit does not contain non-unitary operations like measurement or qubit reset.

Again we use the quantum circuit *circ* from the previous section 4.2, run it with the unitary simulator and then get the unitary for the circuit:

```
from qiskit import Aer

backend = Aer.get_backend('unitary_simulator')
job = backend.run(circ)
unitary = job.result().get_unitary(circ, decimals=3)
print(unitary)
```

We can convert the output to LaTeX by

```
from qiskit.visualization import array_to_latex
print(array_to_latex(unitary).data)
```

As the aim of the circuit is only to prepare the state $|\Psi^-\rangle$ from the initial state $|00\rangle$, here only the first column of the unitary is relevant:

$$\begin{bmatrix} 0 & 0 & 0.707 & 0.707 \\ -0.707 & 0.707 & 0 & 0 \\ 0.707 & 0.707 & 0 & 0 \\ 0 & 0 & -0.707 & 0.707 \end{bmatrix} \tag{14}$$

Aer simulator

The *aer_simulator* is used to simulate a quantum computer, so to get results we have to add measurements to the quantum circuit.

```
from qiskit import Aer

circ.measure_all()
backend = Aer.get_backend('aer_simulator')
job = backend.run(circ, shots=1024)
counts = job.result().get_counts(circ)
print(counts)
```

The *measure_all* method adds a Z-basis measurement on every qubit and creates a classical register for each qubit to store the result. The option `shots` gives the number of times this circuits is run on the backend. If no number of shots is given when running the job, the default of 1024 is selected.

Finally we get the number of counts as a python dictionary, for the example here

```
{'01': 527, '10': 497}
```

We can also visualize the counts in a histogram using the *plot_histogram* function.

```
from qiskit.visualization import plot_histogram
plot_histogram(counts).show()
```

4.4 Measurements and state tomography

In the last example of the previous section we introduced measurements. In this section we have a detailed look at measurements, considering measurements in arbitrary basis and the approximation of the density matrix by state tomography.

We start with a measurement in X-basis. As Qiskit's measurements are always in the Z-basis, we have to implement the X-basis measurement by rotating from the X-basis to the Z-basis in the Bloch sphere. This is achieved by a $-\pi/2$ rotation around the y-axis of the Bloch sphere, which is given by the gate

$$R_y\left(-\frac{\pi}{2}\right) = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}\frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} = ZH\,. \tag{15}$$

So we can implement a X-basis measurement by a Hadamard gate followed by a Z gate and Z-basis measurement. But as the Z here has no influence on the measurement result, we can drop it and just use the Hadamard gate. This is used by convention for the X-basis measurement.

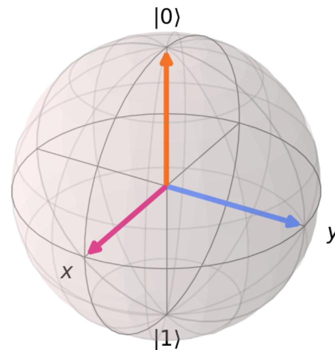

Figure 7: The Bloch sphere with the x, y and z axis marked.

For the Y-basis measurement we proceed similarly. We transform the state from the Y-basis to the X-basis by applying a $\pi/2$ rotation around the z-axis of the Bloch sphere. This transformation is given by the S^+ gate.Then, using the X-basis measurement, the

Y-basis measurement is given by the S^+ gate followed by a Hadamard gate and a Z-basis measurement. For measurements in any other basis one can proceed analogously.

(a) X-basis measurement (b) Y-basis measurement

Figure 8: Implementation of the measurement in X- and Y-basis.

Quantum state tomography [26, 27] can be used to reconstruct the density matrix ρ of a state from a set of informationally complete measurements. This is a set of measurements which allows us to predict the measurement outcome of any other observable. The smallest set of measurements/operators which fulfill this is any basis of the traceless Hermitian operators.

As an example we look at state tomography for a single qubit system, using the pauli basis X, Y and Z. Together with $\mathbb{I}$ they form a basis of the Hermitian 2×2 matrices. So knowing the expectation values of X, Y and Z allows us to reconstruct the density matrix ρ:

$$\rho = \frac{1}{2} \left(\mathbb{I} + \mathrm{Tr}(X\rho)X + \mathrm{Tr}(Y\rho)Y + \mathrm{Tr}(Z\rho)Z \right) \tag{16}$$

But we have to beware that this might not be a physical density matrix. While we see that it is Hermitian and $\mathrm{Tr}(\rho) = 1$, it may not be positive semi-definite.

To consider this, one can use the maximum likelihood estimate, which corresponds to solving the constrained least squares minimization problem:

$$\min_{\rho \geq 0} \sum_i \left[\mathrm{Tr}\left(E_j \rho \right) - p_i \right]^2 \tag{17}$$

Here E_j are the projectors onto the eigenspaces for the operators of the chosen measurement basis (e.g. $|0\rangle\langle 0|$, $|1\rangle\langle 1|$, $|+\rangle\langle +|$, $\ldots$). $\mathrm{Tr}\left(|E_j\rangle\langle E_j| \, \rho \right)$ gives the measurement probabilities for the density matrix ρ. So the density matrix ρ, which fits as good as possible to the statistics of measurement outcomes p_i from the experiments, is chosen by solving the minimization problem.

The big problem of complete state tomography is scalability. For the single qubit we need 3 measurements. As the measurement basis for n qubits is the tensor product of the single qubit measurement basis, we there need 3^n measurements.

We now look at a small example in Qiskit, using the quantum circuit from section 4.2 which prepares the Bell state $|\Psi^-\rangle$.

```
from qiskit_experiments.library import StateTomography

tomo = StateTomography(circ)  # initiate StateTomography instance
tomo.set_run_options(shots=5000)  # set number of shots
job = tomo.run(backend)  # run on a backend
result = job.analysis_results()  # process measurement results
dm = result[0].value  # get density matrix
print(np.round(dm.data, decimals=3))
```

To do state tomography, we first need the `qiskit_experiments` package. Then we can create a `StateTomography` instance. This contains all the relevant parameters, here we just provide the circuit, as we want to do full state tomography on all qubits. We could also do state tomography for just a subset of our circuits qubits (`measurement_qubits`) or only do partial state tomography.[13]

We adjust the number of shots and run the state tomography on a simulator or real quantum computer and finally process the results to get the density matrix. Running this on the Aer simulator gives:

$$\rho = \begin{bmatrix} 0 & -0.002-0.004i & 0.002+0.004i & 0 \\ -0.002+0.004i & 0.492 & -\frac{1}{2} & 0.002+0.004i \\ 0.002-0.004i & -\frac{1}{2} & 0.508 & -0.002-0.004i \\ 0 & 0.002-0.004i & -0.002+0.004i & 0 \end{bmatrix} \tag{18}$$

At the end of the next section 4.5 we see the result from a real device.

4.5 Running a quantum circuit on the IBM hardware

Running a quantum circuit on the IBM hardware is similar to running it on the *aer_simulator*. We again look at a short example using the quantum circuit *circ* from section 4.2.

```
1  from qiskit import IBMQ
2  from qiskit.compiler import transpile
3
4  #login to IBMQ
5  IBMQ.save_account('API token')  # only needed if first login
6  IBMQ.load_account()
7  provider = IBMQ.get_provider(group='open')
8
9  circ.measure_all()
10 backend = provider.get_backend('ibmq_quito')
11 circ_t = transpile(circ, backend, optimization_level=0)
12 job = backend.run(circ_t, shots=1024)
13 counts = job.result().get_counts(circ_t)
14 print(counts)
```

First we login to the ibmq network. The API token can be found at the account settings page[14]. It is not needed if the IBM Jupyter notebook is used.

Then we need to choose a backend, instead of manually selecting a backend we can also use the *least_busy* function for finding the device with the lowest number of pending jobs out of the list of devices which have enough qubits to run our circuit.

```
1  from qiskit.providers.ibmq import least_busy
2
3  num_qubits = 2
```

[13] For doing partial state tomography, one specifies with the parameter `basis_indices` which measurements should be applied. For example if we would want to measure only in the $Z \otimes Z$, $X \otimes X$ and $Y \otimes Y$ basis we would use `[[0, 0], [1, 1], [2, 2]]`.

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

```
4 backend = least_busy(provider.backends(filters=lambda x: x.configuration
    ().n_qubits >= num_qubits and not x.configuration().simulator and x.
    status().operational), reservation_lookahead=60)
```

The reservation_lookahead option is sorting out backends with reservations for the next n minutes.

If we want to find the best possible backend for running a specific circuit, it makes sense to consider the qubit connectivity and error map (see Figure 2 for an example):

```
1 from qiskit.visualization import plot_error_map
2 plot_error_map(backend).show()
```

Before running the circuit on the selected backend the circuit has to be processed by the *Transpiler*, which adjusts and optimizes the quantum circuit for the given backend. We look in detail at this in the next section.

Then we send the circuit to the quantum computer and have to wait for the result. Finally we access the counts for the experiment, here are as an example counts for our circuit which prepares the Bell state $|\Psi^-\rangle$:

```
1 {'00': 82, '01': 466, '10': 447, '11': 29}
```

On an ideal device we should only get counts for '01' and '10', the counts for '00' and '11' have to be caused by noise.

Running state tomography example from last section again on the *ibmq_manila* gives the following density matrix:[15]

$$\begin{bmatrix} 0.08743 & -0.00532-0.0081i & 0.04008-0.03498i & -0.00956-0.0043i \\ -0.00532+0.0081i & 0.48406 & -0.38188+0.01234i & 0.04532-0.0305i \\ 0.04008+0.03498i & -0.38188-0.01234i & 0.40449 & -0.00848+0.00066i \\ -0.00956+0.0043i & 0.04532+0.0305i & -0.00848-0.00066i & 0.02401 \end{bmatrix} \tag{19}$$

To compare this result with the ideal or simulation result we look at the fidelity. This is defined as

$$F(\rho_1,\rho_2) = \left[\mathrm{Tr}\left(\sqrt{\sqrt{\rho_1}\rho_2\sqrt{\rho_1}}\right)\right]^2 \tag{20}$$

and gives 1 if the corresponding states are equal and 0 if they are orthogonal. In Qiskit it can be computed by

```
1 from qiskit.quantum_info import state_fidelity
2 print(state_fidelity(dm, dm_sim))
```

Here we get a fidelity of 0.826 comparing with the ideal result. We will see in section 4.7 how to improve this.

[15] Circuit run on 01.08.2022 with *ibmq_manila*.

If one wants to access the results of a previously run circuit this is possibly by using its job id[16]

```
1 job = backend.retrieve_job('job_id')
```

On the IBM devices it is currently not possible to run a circuit with gates that are conditionally implemented based on previous measurement results (which is available in the simulator using the *c_if* method of the gate objects), but this is planned in the road map for 2022 [28].

4.6 Qiskit Transpiler

In the current IBM quantum computers each qubit is only connected via CNOT gates to a few other qubits (see e.g. Figure 2). Thus we can not directly run an arbitrary quantum circuit, as this would very likely contain two-qubit gates applied on pairs of two hardware qubits which are not connected.

For looking at this problem we introduce a few definitions.

- The qubits our quantum circuit is build on, so the 'qubits' we created in Qiskit by either creating a `QuantumCircuit` or `QuantumRegister` we call *virtual qubits*.
- The mapping π is the mapping of the virtual qubits to *hardware qubits*. This mapping is bijective if we add as many ancilla qubits to the set of virtual qubits as we have more hardware qubits then virtual ones.
- The metric $D(\pi(q_1), \pi(q_2))$ is the length of the shortest connection path between the hardware qubits $\pi(q_1)$ and $\pi(q_2)$. Thus $D(\pi(q_1), \pi(q_2)) = 1$ exactly if a CNOT can be implemented between the hardware qubits $\pi(q_1)$ and $\pi(q_2)$.
- We call a two-qubit gate $g(q_1, q_2)$ directly executable if $D(\pi(q_1), \pi(q_2)) = 1$.
- If there exists a mapping π, which makes all two-qubit gates directly executable, we call it a perfect mapping.

For (very) simple circuits one might find a perfect mapping π. But usually this is not possible. If - while executing the circuit - we have to implement a gate $g(q_1, q_2)$ that is not directly executable, we have to modify the mapping π in order that the assigned hardware qubits $\pi(q_1)$ and $\pi(q_2)$ are connected. In the quantum circuit this is implemented by adding two-qubit SWAP gates where necessary. Finding the optimal solution here is a NP-hard problem [29], so heuristic algorithms have to be used to find a good enough solution.

Also the gates in the circuit have to be decomposed into the supported basis gates of the quantum computer. Optionally the circuit can be further optimized by combining gates, e.g. if we have a chain of two self-adjoint gates they can be canceled.

The Qiskit *Transpiler* offers some tools for these tasks. It works by generating a *pass manager* based on the given arguments (e.g. `optimization_level`). This *pass manager*

[16]The job id can be found in the list of previous jobs at `https://quantum-computing.ibm.com/jobs`

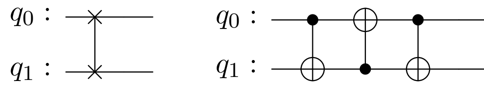

Figure 9: On the IBM devices a swap gate is implemented using three CNOT gates.

then manages the execution of the individual circuit transformations (*passes*). These transformations for example transform the quantum gates into the basis gates of the backend, add swap gates into the circuit in order to adapt it to the qubit layout of the device or reduce the amount of gates used by combining gates.

Here we only look at some of the *transpile* functions optional arguments and the corresponding algorithms, how to create custom *pass managers* and *passes* can be found in the Qiskit documentation.[17]

For simple circuits it is sufficient to provide the `optimization_level=n` argument, where n can range from 0 to 3, to choose one of the predefined settings. The different settings consist of:

n=0 is mapping the virtual qubits to the physical qubits with same index ($\pi = id$).

n=1 chooses $\pi = id$, if then all two-qubit gates are directly executable. Otherwise the qubit layout is chosen in the same way as for $n = 2$. Additionally chains of single qubit unitary gates are combined and groups of two neighboured CNOTs with same target and control are canceled. Also unneeded resets, e.g. at the beginning of the circuit are removed.

n=2 first tries to find a perfect mapping π. If multiple perfect mappings are found, the least noisy one is chosen by considering the average gate fidelities of the device weighed by the number of gates applied on these qubits/two-qubit connections. If this is not found `layout_method='dense'` is used. Again chains of single qubit unitary gates are combined and redundant resets are removed. Also the circuit is analyzed for sub-circuits of commuting gates and then in these sub-circuits multiple of 2 occurrences of the identical (also same qubit arguments) self-adjoint gate are deleted.

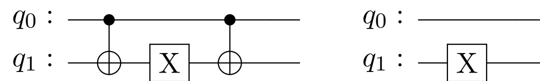

Figure 10: Simple example for commuting gates optimization.

[17] Advanced circuits/Transpiler passes and pass manager in [23]

n=3 first tries to find a perfect mapping in the same way as for $n = 2$. If this is not found, then `layout_method='sabre'` is used. In addition to the circuit optimizations from $n = 2$, two-qubit subcircuits are re-synthesized by converting them to a unitary matrix and then decomposing this matrix into the quantum computers basis gates again. Also one- and two-qubit gates which are represented by a diagonal unitary matrix, like R_z, T, Z, CZ, etc., are removed if followed by measurements.

For inserting the necessary SWAPs Qiskit first converts the circuit into a Directed Acyclic Graph (DAG). Each node of the graph represents a two-qubit gate[18] and the edges visualize the dependencies between the two-qubit gates (see Figure 11 for an example). One defines the first layer L_1 of the DAG as the set of nodes which do not depend on

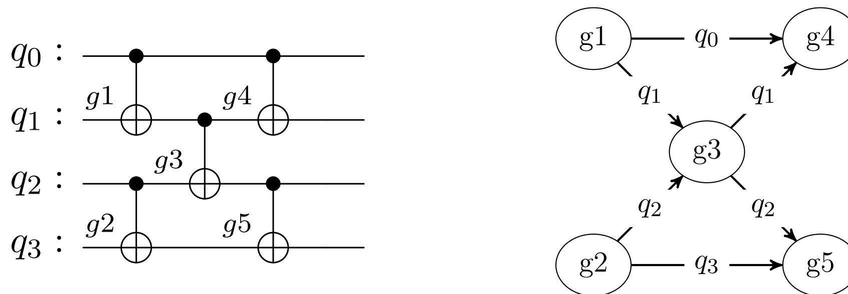

Figure 11: Example for converting a quantum circuit into a Directed Acyclic Graph (DAG).

other nodes and iteratively the n-th layer L_n as the set of nodes which only depend on the previous $n-1$ layers.

The optional argument `routing_method` defines the algorithm by which SWAP gates are inserted. The default setting is `'stochastic'`, except for `optimization_level=3` where it is `'sabre'`. Given an initial mapping π_i, here π is the modified mapping at the beginning of each iteration step.

'basic' The simplest routing method just iterates over the layers and for each layer over the two-qubit gates. If a gate $g(q_1, q_2)$ is not directly executable, SWAPs are inserted according to the shortest connection between the hardware qubits $\pi(q_1)$ and $\pi(q_2)$.

'stochastic' Iteratively for each layer L_k of the circuits DAG the following random algorithm is run 20 times.[19]

1. Let n be the number of hardware qubits. A symmetric $n \times n$ random matrix S is generated, its entries are sampled by the normal distribution with mean

[18]Here we ignore single qubit gates, as well as input and output nodes. In Qiskit the single qubit gates are represented in the DAG as nodes with only one input and one output edge. The input nodes have only one outgoing edge and represent the initialization of the device in the $|0\rangle$ state. The output nodes only one incoming edge and represent measurements.

[19]If run with `optimization_level=3` 200 times.

1 and standard deviation $1/n$.

$$S_{ij} = S_{ji} \sim \mathcal{N}\left(1, \frac{1}{n^2}\right) \tag{21}$$

Then the cost function is defined by

$$c_r = \sum_{g(q_1,q_2)\in L_k} D(\pi(q_1), \pi(q_2)) S_{\pi(q_1),\pi(q_2)} \tag{22}$$

2. Now it is seared for a combination of swaps which maps any two virtual qubits of a two-qubit gate in the current layer to two connected hardware qubits.
   a) If a swap of two of the circuit's virtual qubits with $D(\pi(q_1), \pi(q_2)) = 1$ reduces the cost function, it is implemented. These two qubits are not considered for swapping in this step anymore.
   b) If there are no swap pairs left which reduce the cost, then for the next depth layer of swap gates one goes to the previous step again. This is repeated until a solution has been found. If within $2n$ depth layers no solution is found the trial has failed.

If multiple solutions are found, then the solution with the least depth layers is used. If no solution is found, one proceeds for this layer gate by gate.

'sabre' This implements the routing algorithm from [30]. The algorithm iterates though the circuits DAG in the following steps:

1. If there are directly executable gates in the first layer L_1 of the DAG, then these are appended to a list[20] and removed from the DAG. This is repeated until no directly executable gates are left in the first layer.
2. We define Q_1 as the set of virtual qubits, where a gate from the first layer is applied on. Then

$$S = \{(q, w) | q \in Q_1,\ D(\pi(q), \pi(w)) = 1\} \tag{23}$$

   is the set of two-tuples of virtual qubits (q, w), where $q \in Q_1$ and the corresponding hardware qubits $\pi(q)$ and $\pi(w)$ are connected via a CNOT.
3. A heuristic cost function is used to rate each mapping π_{new} which results of a swapping the qubits from a tuple in S. In Qiskit there are three cost functions implemented[21]:

   'basic'

$$c_1 = \sum_{g(q_1,q_2)\in L_1} D(\pi_{\text{new}}(q_1), \pi_{\text{new}}(q_2)) \tag{24}$$

   The sum of the distances of the hardware qubits corresponding to the virtual qubits of each gate in the first layer.

[20] From this list at the end a new circuit can be created.

[21] It depends on the chosen `optimization_level`, which cost function is used: `'basic'` for 0, `'lookahead'` for 1 and `'decay'` for 2 and 3.

'lookahead'

$$c_2 = \frac{1}{|L_1|} \sum_{g(q_1,q_2)\in L_1} D(\pi_{\text{new}}(q_1), \pi_{\text{new}}(q_2)) + \frac{W}{|L_2|} \sum_{g(q_1,q_2)\in L_2} D(\pi_{\text{new}}(q_1), \pi_{\text{new}}(q_2)) \tag{25}$$

The average of the distances of the hardware qubits corresponding to the virtual qubits of each gate in the first two layers, where the average for the second layer is weighted by $W \leq 1$.[22]

'decay' In order that the applied SWAPs can be executed in parallel as much as possible, we define the function swap_counter(h) as a counter which counts the number of swaps the hardware qubit h took part in. Then the *decay* function is then defined to be the maximum of the swap_counter for the hardware qubits on which the virtual qubits considered for swapping are mapped:

$$\text{decay}(q, w) = \max\left[\text{swap_counter}(\pi(q)), \text{swap_counter}(\pi(w))\right] \tag{26}$$

The cost function used here is then given by

$$c_3(q, w) = [1 + \delta \text{decay}(q, w)]\, c_2\,, \tag{27}$$

where c_2 is the cost function from `'lookahead'`. In Qiskit $\delta = 0.001$ is used and the swap_counter is reset to zeros every 5 iterations of the algorithm.

Finally a SWAP gate is applied to the two-qubit connection with the lowest cost score and one starts at step 1 again.

When no node is remaining in the DAG, we have found a set of SWAPs which makes the circuit compatible with the hardware.

'lookahead' This algorithm [31] starts similar as the previous one. The first two steps are the same, in the third step - instead of only choosing one mapping - the n mappings π_{new} with the lowest score by the cost function c_1 are selected.

For all n new mappings these steps are repeated, until we have reached m iterations. So finally we have n^m combinations of m SWAPs. Out of these, the combination which allows to execute the most two-qubit gates, is implemented.[23]:

'toqm' This option requires the `qiskit-toqm` package. It implements the *Time-Optimal Qubit Mapping* [32]. This swap insertion scheme prefers qubits with faster gate execution times and thus optimizes the circuit execution time. By a faster circuit execution time the effect of qubit decoherence over time is reduced.

[22] It is not necessary to consider all gates from the second layer, as - due to more SWAPs likely to be inserted until these gates are executed - this can only be considered as an inaccurate estimation of the effect of a SWAP. In Qiskit `0.37.0` a maximum of 20 gates from the second layer are considered.

[23] The values of (n, m) depend on the chosen `optimization_level`:
$(2, 2)$ for 0, $(4, 4)$ for 1, $(5, 5)$ for 2 and $(6, 5)$ for 3.

The result of the SWAP mapping can be improved by choosing a good initial mapping π_i of virtual qubits to physical ones.

The method by with the initial mapping π_i is found can also be specified individually by the optional argument `layout_method`. By specifying this, the search for a perfect mapping will be omitted, and only the specified layout method used. Possible options are:

'trivial' The virtual qubits are mapped to the physical qubits with same index ($\pi_i = id$).

'dense' The virtual qubits are mapped to the best found connected subgroup of physical qubits for the given circuit. For a circuit with n qubits, first it is searched for the subgroups of n physical qubits with the highest number of connections to other physical qubits in the subgroup. If here more than one best connected subgroup is found, the average CNOT and average measurement error, weighted by the number of CNOTs and measurements in the circuit is used to select the subgroup used.

'noise_adaptive' This implements the *Greatest Weighted Edge First (GreedyE)* mapping proposed in [33]. First the circuit is used to create a *program graph* where the nodes represent the virtual qubits and edges represent two-qubit gates applied on those. The number of two-qubit interactions between two nodes is assigned as a weight to the corresponding edge. The CNOT error rates of the backend are used to calculate the SWAP reliability for SWAP gates between any two hardware qubits.[24]

The qubit mapping starts by placing the highest weighted edge of the program graph at the best hardware CNOT connection Iteratively, the highest weighted program edge which has only one endpoint assigned is chosen. It's unassigned node is mapped to the hardware qubit with the highest product of readout reliability and total CNOT reliability to the hardware qubits corresponding to already mapped neighboured program nodes (using the initially computed SWAP reliabilities).

'sabre' This implements the SABRE layout mapping from [30]. Starting with a random initial mapping π_i, one iterates these steps:

1. The SABRE routing method is applied for the given circuit and initial mapping π_i to find a set of SWAP gates. But instead of inserting the SWAP gates into a quantum circuit we are only interested in the final mapping π_f. This is set as the new initial mapping π_i.
2. The inverse of the circuit is calculated and the same procedure applied to it. We thus get an updated initial mapping.

In Qiskit the number of iterations depends on the chosen `optimization_level`.[25]

[24]This is implemented by assigning the logarithm of the SWAP reliability for any two connected hardware qubits as weights to a graph and then computing the weight of the shortest path using the Floyd–Warshall algorithm.

[25]This is specifically one for $n = 0$, two for $n = 1$ and $n = 2$ and four for $n = 3$.

The initial mapping π can also be provided manually by using the parameter *initial_layout*. One of the supported formats is a list of physical qubit numbers, which has to be ordered in the same way as the corresponding virtual qubits.

The method used to convert the circuit gates into basis gates of the quantum computer is set by the optional argument `translation_method`:

'unroller' Non-basis gates are expanded into basis gates U_1, U_2, U_3 and CNOT according to a predefined rule set. For a different set of basis gates this might fail.

'translator' First approximates any unitary matrix gates in the circuit with basis gates and decomposes custom gate definitions.[26] Then generates a rule set for the translation into a given basis. This is based on an equivalence library, which contains decompositions for all predefined gates in Qiskit. This is the *default option*.

'synthesis' This maps any gate to basis gates by converting them to unitary matrices and then approximates the unitary matrices with basis gates. One- or two-qubit subcircuits are converted into a single unitary matrix first.

This does not work with the non-unitary initialise instruction and with custom gates, if they are composed of other gates instead of defined by a unitary matrix.

The `approximation_degree` sets the level of approximation applied every time when approximating unitary matrices with quantum gates. The value can be set in the interval $(0, 1]$, where 1 is the default value and corresponds to minimal approximation.

The last parameter of the Transpiler we look at is `callback`. Here we can provide a callback function to the Transpiler, which lets us monitor or even modify the results of individual transpiler passes.

For a simple example count the number of SWAP gates inserted into the circuit before they are decomposed into CNOT gates:

```
1  from qiskit import QuantumCircuit, transpile
2
3  def simple_callback(**kwargs):
4      pass_ = kwargs['pass_']  # pass that was executed
5      dag = kwargs['dag']  # DAG output of the pass
6      time = kwargs['time']  # time needed to execute the pass
7      swap = ['BasicSwap', 'LookaheadSwap', 'StochasticSwap', 'SabreSwap']
8      if pass_.name() in swap:
9          swap_count = len(dag.named_nodes('swap'))  # count swap nodes
10         print(f'{pass_.name()}: {swap_count} SWAPs inserted in {time} s')
11
12 circ = QuantumCircuit(7)
13 for i in range(1, 7):
14     circ.cx(0, i)
15
16 c_map = [[0, 1], [1, 0], [1, 2], [1, 3], [2, 1], [3, 1], [3, 5], [4, 5],
       [5, 3], [5, 4], [5, 6], [6, 5]]  # ibm_nairobi coupling map
17 circ_t = transpile(circ, coupling_map=c_map, callback=simple_callback)
```

[26]For details about unitary and custom gates see section 4.8.

We use the `coupling_map` instead of a `backend` for keeping this example minimal. The format of it is a list which contains lists of the structure [control_qubit, target_qubit] for each two qubit CNOT interaction of the quantum computer.

In the callback function `simple_callback` we first access the variables passed by the Transpiler. Two more available variables which are not used in this example are `kwargs ['property_set']`, which contains information about the circuit like the initial and final mappings π, and the index of the pass `kwargs['count']`. We then filter for one of the four routing passes, which are explained on the previous pages, and print the name and execution time of the pass as well as the number of SWAP gates inserted in the quantum circuit. [27]

Instead of just counting gates we could also modify the DAG here:

```
1 from qiskit.converters import circuit_to_dag
2
3 circ = ...  # define the new node by its QuantumCircuit
4 new_dag = circuit_to_dag(circ)  # convert QuantumCircuit to dag
5 # call in callback function to substitute first swap node with new_dag:
6     node = dag.named_nodes('swap')[0]
7     dag.substitute_node_with_dag(node, new_dag)
```

We do not even have to define the circuit modification in the dag representation, as we can let Qiskit convert it.

This finishes the section about the Transpiler, a list of all options can be found in the Qiskit documentation.[28]

4.7 Readout error correction

Looking at the results from section 4.5 (see blue histogram in Figure 12) we see that we do not get the counts we would expect for an ideal device - e.g. the state $|00\rangle$ or $|11\rangle$ should not be measured when we prepare the state $|\Psi^-\rangle$. So now we look at a simple error correction method, which is already implemented in Qiskit.

There are different kinds of noise one has to consider when running a circuit on a quantum computer. Here we only look at the readout noise, which is the noise introduced during the measurement process, as this can be mitigated using classical post processing without needing additional qubits. Also for circuits with few gates this is the dominant noise.

The concept is to prepare and measure every basis state many times and then use the measurement outcomes to create a matrix Λ which approximates the ideal measurement statistics for an arbitrary state, given the noisy ones. For example if we prepare $|11\rangle$ and the result for 1000 measurements is

$$\{\text{'01': 10, '10': 10, '11':980}\} \tag{28}$$

then

$$\begin{pmatrix} 0 & 0.01 & 0.01 & 0.98 \end{pmatrix}^T \tag{29}$$

[27] If we used a circuit which is already defined with some SWAP gates, the would be converted into basis gates before calling the routing pass and thus not counted here.

[28] https://qiskit.org/documentation/stubs/qiskit.compiler.transpile.html

gives the last column of our matrix Λ.

Given the counts of the outcomes from noisy measurements, we can reconstruct the ideal counts by the inverse of Λ

$$\begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix}_{\text{ideal}} = \Lambda^{-1} \begin{pmatrix} c_{00} \\ c_{01} \\ c_{10} \\ c_{11} \end{pmatrix}_{\text{noisy}} \tag{30}$$

where c_{00}, c_{01}, c_{10} and c_{11} are the counts of the corresponding outcomes.

Now we look at how this procedure is implemented in Qiskit:

```
1 from qiskit.utils.mitigation.circuits import complete_meas_cal
2 from qiskit.utils.mitigation.fitters import CompleteMeasFitter
3
4 # backend = ...
5 meas_calibs, state_labels = complete_meas_cal(range(5), circlabel='mcal')
6 cal_job = backend.run(meas_calibs, shots=8192)
7 cal_results = cal_job.result()
8 meas_fitter = CompleteMeasFitter(cal_results, state_labels, circlabel='
    mcal')
9 print(meas_fitter.cal_matrix)
```

In line 4 the calibration circuits which prepare and measure all the 2^n basis states are created by the *complete_ meas_ cal* function, so *meas_ calibs* is a list of 2^n circuits and the list *state_ labels* contains the corresponding state labels.
On the IBM machines these circuits can be run directly, as they only contain measurements and the Pauli X gate, which is one of the basis gates for these devices. If using Qiskit on other devices which do not have the Pauli X as a basis gate, we have to use the transpiler to convert it into basis gates:

```
1 meas_calibs = transpile(meas_calibs, backend=backend, optimization_level
    =0)
```

It is important that *optimization_ level* is not set higher than 1, as the qubit labels must not be changed for the calibration circuits.
Using *CompleteMeasFitter* one finally gets the calibration matrix Λ.
Looking at the reduced calibration matrices, e.g. for the first qubit, we see that the readout noise mainly consists of decay from the excited state $|1\rangle$ to the ground state $|0\rangle$.

$$\Lambda_0 = \begin{pmatrix} 0.98203 & 0.08313 \\ 0.01797 & 0.91687 \end{pmatrix} \tag{31}$$

The values on the diagonal of Λ_0 are the probabilities for a state prepared in $|0\rangle$ ($|1\rangle$) to be measured in '0' ('1'). 0.08313 is the probability for a state to be measured in '0' if prepared in $|1\rangle$ and 0.01797 is the probability for a state to be measured in '1' if prepared in $|0\rangle$. For non-basis states this then corresponds to getting measurement outcome '0' 8.313% of the times the outcome of the ideal measurement would have been '1' and 1.797% for vice versa.

The ideal measurement can now be predicted by calculating Λ for the subsystem of the qubits used (we could also initially have just created the calibration circuits for the first 2 qubits). Then Qiskit can apply Λ^{-1} on the noisy measurement result.
We again use the quantum circuit *circ* from section 4.2. When transpiling the circuit it is important that we use *layout_ method='trivial'*, in order that the circuit is run on the assigned qubits.

```
1 meas_fitter_2 = meas_fitter.subset_fitter([0, 1])
2 meas_filter_2 = meas_fitter_2.filter
3
4 circ_t = transpile(circ, backend, optimization_level=3, layout_method='
      trivial')
5 job = backend.run(circ_t, shots=8192)
6 counts = job.result().get_counts(circ)
7
8 corr_counts = meas_filter_2.apply(counts)
```

If both better optimisation and readout correction is needed, one could first transpile the circuit and then apply measurements.

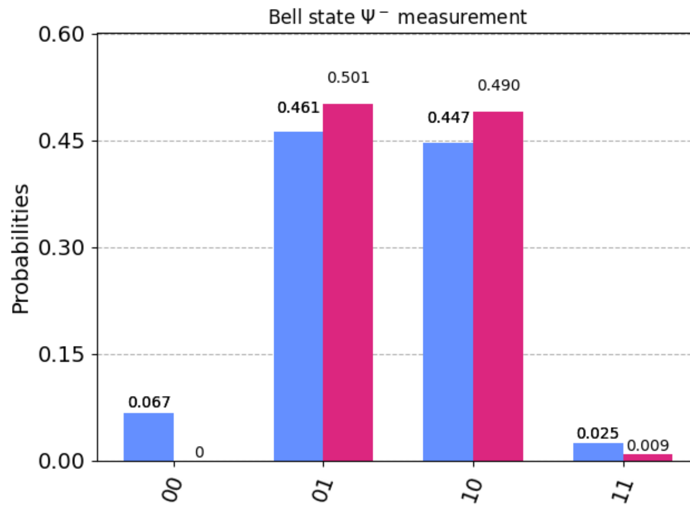

Figure 12: Measurement results for prepared state $|\Psi^-\rangle$, without (blue) and with (red) readout error correction

We need 2^n measurements for n qubits, so this error correction procedure is not well scalable. But if we assume that the correlation between readout errors of different (groups of) qubits can be neglected, we can use use *tensored_ meas_ cal* and *TensoredMeasFitter*, which creates the calibration matrices for single qubits (or groups of qubits) like proposed in [34]. This only uses 2^p measurements, where p is the size of the largest group of qubits where we do want complete readout tomography.

```
1 from qiskit.utils.mitigation.circuits import tensored_meas_cal
2 from qiskit.utils.mitigation.fitters import TensoredMeasFitter
```

```
# backend = ...
meas_calibs, state_labels = tensored_meas_cal(mit_pattern=[[0, 1], [2],
    [3, 4]], circlabel='mcal')
cal_job = backend.run(meas_calibs, shots=8192)
meas_fitter = TensoredMeasFitter(cal_job.result(), state_labels,
    circlabel='mcal')
# counts = {'00000': 234, ...}
corr = meas_fitter.filter.apply(counts)
```

The main change (besides replacing complete with tensored) is that in line 5 we give the groups of qubits instead of just a list of qubit numbers. This example would only use 4 instead of 32 measurements.

We now look if this can also improve our state tomography example from the end of section 4.5 and run this again with applying readout error correction. We get the density matrix[29]

$$\begin{bmatrix} 0.00751 & -0.00786+0.00918i & 0.00445-0.00391i & -0.00488-0.00316i \\ -0.00786-0.00918i & 0.49394 & -0.48867+0.01316i & 0.01322+0.0096i \\ 0.00445+0.00391i & -0.48867-0.01316i & 0.49335 & -0.01279-0.00629i \\ -0.00488+0.00316i & 0.01322-0.0096i & -0.01279+0.00629i & 0.00519 \end{bmatrix} \tag{32}$$

and a fidelity to the ideal density matrix of 0.982, which is significantly improved in comparison to results without error mitigation eq. (19).

4.8 Advanced gates

In this section we will look at how to simplify the creation of quantum circuits by using custom gates and parameterized gates.

If we want to add a one- or two-qubit gate, where we know its representation as a unitary matrix, to our circuit, we can just append this matrix to our circuit. The transpile function will convert the unitary into the given basis gates.

```
from qiskit import QuantumCircuit, quantum_info

circ = QuantumCircuit(2)
unitary = quantum_info.random_unitary(4)
circ.unitary(unitary, [0, 1])
```

Up to two-qubits this is quite efficient, needing 3 CNOT gates for random two-qubit unitaries. But trying to decompose 3-qubit or even larger unitaries will lead to a very long circuit.

So in this case we might want to define custom gates by decompositions in one- and two-qubit Qiskit gates or unitaries:

```
from qiskit import QuantumCircuit, quantum_info

def_circ = QuantumCircuit(3)
def_circ.unitary(quantum_info.random_unitary(4), [0, 1])
```

[29]Circuit run on 01.08.2022 with *ibmq_ manila*.

```
def_circ.cx(0, 2)
gate = def_circ.to_instruction()

new_circ = QuantumCircuit(3)
new_circ.append(gate, range(3))
```

The `.to_instruction()` creates a custom gate from the circuit its called on. This is then added to another circuit by `.append(gate, qubits)`.

For creating many similar circuits Qiskit offers parameterized gates. We give a short example by adding one more gate to the previous circuit `new_circ`:

```
from qiskit.circuit import Parameter
import numpy as np

theta = Parameter('theta')
new_circ.rx(theta, 0)
circuits = new_circ.bind_parameters({theta: np.pi/4})
```

Usually we would apply the transpile function before assigning a set of parameters. Then the circuit has only to be compiled once for all of the different parameters it should be run with. This reduces the compilation time, which becomes relevant when compiling large circuits. More about parameterized gates can be found in the first part of the advanced circuits section in [23].

5 Case study: Preparation of mixed Werner state and Bell diagonal states

The two-qubit Werner state is a convex combination of the maximally maximally mixed state $\frac{\mathbb{I}_{4\times 4}}{4}$ and the bell state $|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle)$:

$$W_p = (1-p)\frac{\mathbb{I}_{4\times 4}}{4} + p\left|\Psi^-\right\rangle\left\langle\Psi^-\right| \qquad p \in [0,1] \tag{33}$$

Notice that depending on p one gets a maximally entangled bell state to a maximally mixed state. One can show with the PPT criterion that this state is entangled for $p > \frac{1}{3}$.

In this section we want to prepare a Werner state on an IBM quantum computer. To gain some intuition, we first look at the simple approach of simulating the Werner state by preparing the two-qubit maximally mixed state $\frac{\mathbb{I}_{4\times 4}}{4}$ and the bell state $|\Psi^-\rangle$ separately. After that we discuss a more general and realistic method from [35] to prepare Bell Diagonal states.

5.1 Simulating a Werner state using classical post-processing

We start with discussing a naive, natural scheme of preparing the Werner state. Unfortunately this cannot be realised in a IBM quantum computer, since the use of gates conditioned on certain measurement outcomes on other qubits is not yet allowed. Implementing these conditional gates is in IBM's road map and expected to be realized soon [].

First we need to prepare a bell state and a maximally mixed state. For preparing the bell state we use our quantum circuit from section 4.2:

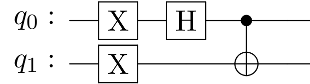

Figure 13: The circuit used to prepare a $|\Psi^-\rangle$ state.

The maximally mixed state can be prepared by using a Hadamard gate followed by a measurement. The result of it is discarded, so after the measurement we have the one-qubit maximally mixed state $\mathbb{I}_{2\times 2}/2 = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$.

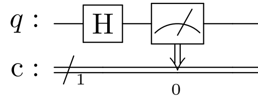

Figure 14: Preparing a mixed state

Now we combine these to one circuit that we can run on the IBM quantum computers to simulate a Werner state. In order to need one qubit less we only prepare a one-qubit maximally mixed state. The two qubit maximally mixed state is then given by measuring

the one-qubit state and one qubit of the Bell state. We can do this, as the Bell state is maximally entangled, so the reduced density matrix for one qubit is $\mathbb{I}_{2\times2}/2$.

Finally we add on q_3 a $R_Y(2\arccos(\sqrt{p}))$ gate which prepares the single qubit state $\sqrt{p}\,|0\rangle + \sqrt{1-p}\,|1\rangle$. The measurement of this state then by probability p returns the state $|0\rangle$, so based on its result we can select the bell state or the maximally mixed state.

If it would be possible to implement a gate based on the result of a previous measurement,[30] we could modify the circuit to prepare a Werner state on the quantum computer by adding a SWAP gate, which swaps q_1 and q_2 if $c_3 = 1$, which is the case exactly if q_3 is measured in state $|1\rangle$. Then we would have a two-qubit Werner state prepared on q_0 and q_1 with this circuit:

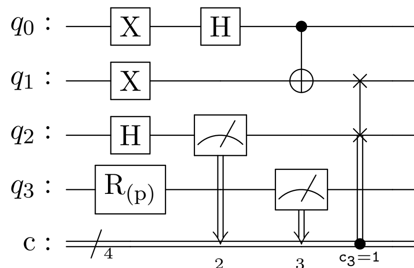

Figure 15: A circuit to prepare the Werner state on an IBM quantum computer using a conditional swap gate, which cannot be implemented currently. The conditional swap gate is the last gate of the circuit, it is only implemented if $c_3 = 1$, which is the case exactly if q_3 is measured in state $|1\rangle$. This circuit can be implemented once IBM allows for the use of gates conditionally implemented based on the state of the classical register, which is expected to be realized soon.

With some classical post-processing it is possible to simulate a Werner state this way, but as we would like to prepare the state in order to also be able to apply further unitary operations on it, this is not very useful.

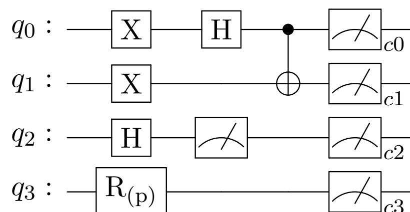

Figure 16: A circuit to simulate the Werner state on an IBM quantum computer. The gate $R(p)$ is given by $R_Y(2\arccos(\sqrt{p}))$.

After running this circuit we then decide by a classical random bit n (which is 0 with

[30] IBM intends to implement this soon [28].

probabilty p) or the measurement of q_3 which two measurement results of c_0, c_1 and c_2 we take. For the mixed state ($n = 1$ or $c_3 = 1$) we take c_0 and c_2 and for the bell state ($n = 0$ or $c_3 = 0$) it's c_0 and c_1.

5.2 Preparing a Werner state using Bell Diagonal states

We now look at a method to prepare a Bell state on a quantum computer with the currently allowed gates using Bell Diagonal states [35, 36].

The class of the Bell-diagonal states (BDS) is given by convex combinations of the 4 Bell states:

$$\left|\Phi^{\pm}\right\rangle = \frac{1}{\sqrt{2}}\left(|00\rangle \pm |11\rangle\right) \text{ and } \left|\Psi^{\pm}\right\rangle = \frac{1}{\sqrt{2}}\left(|01\rangle \pm |10\rangle\right) \tag{34}$$

Thus we can describe a BDS by the 4 parameters $0 \leq p_{jk} \quad i, j \in \{0, 1\}$

$$\rho = \sum_{j,k=0}^{1} p_{jk} \left|\beta_{jk}\right\rangle \left\langle\beta_{jk}\right| \tag{35}$$

where $|\beta_{00}\rangle{=}|\Phi^+\rangle$, $|\beta_{10}\rangle{=}|\Phi^-\rangle$, $|\beta_{01}\rangle{=}|\Psi^+\rangle$, $|\beta_{11}\rangle{=}|\Psi^-\rangle$ and $\sum_{jk} p_{jk} = 0$.

The totally mixed state $\mathbb{I}/4$ is given for $p_{ij} = 1/4$. Then we get the Werner state in the Bell basis given by

$$\rho = (1-p) \sum_{j,k=0}^{1} \frac{1}{4} \left|\beta_{jk}\right\rangle \left\langle\beta_{jk}\right| + p \left|\beta_{11}\right\rangle \left\langle\beta_{11}\right| \quad p \in [0, 1] \ , \tag{36}$$

which corresponds to $p_{00} = p_{01} = p_{10} = \frac{1-p}{4}$ and $p_{11} = \frac{1+3p}{4}$.

In [35, 36] a circuit is proposed for preparing Bell-diagonal states:

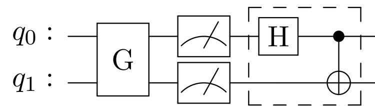

Figure 17: Circuit for preparing Bell diagonal states.

The gate G prepares a two-qubit state with the amplitudes $\sqrt{p_{jk}}$:

$$G\left|00\right\rangle = \sum_{j,k=0}^{1} \sqrt{p_{jk}} \left|jk\right\rangle \tag{37}$$

Now we measure both qubits to project the state into

$$\rho_m = \sum_{j,k=0}^{1} p_{jk} \left|jk\right\rangle \left\langle jk\right| \tag{38}$$

The combination of the Hadarmard and CNOT gate, which we also used in section 4.2, prepares the Bell states $|\beta_{jk}\rangle$ if applied on the pure states $|jk\rangle$. So the state from (38) gets transformed into a BDS:

$$\rho = \sum_{j,k=0}^{1} p_{jk} |\beta_{jk}\rangle \langle\beta_{jk}| \tag{39}$$

To prepare G we use Qiskit's initialize method, which prepares a given state vector (see section 4.2 for how to use this.).

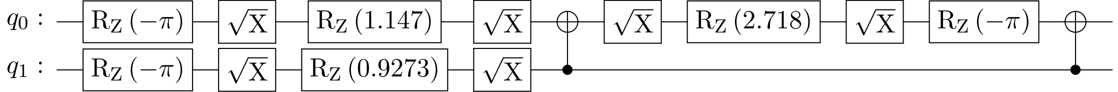

Figure 18: Circuit for preparing the state from eq. (37) for $p = 0.6$ (so $p_{00} = p_{01} = p_{10} = 0.1$ and $p_{11} = 0.7$).

In Table 2 we see the fidelity between the Werner states prepared with this circuit and the ideal states. For $p \geq 0.6$ also readout error correction is applied.

p	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
$F(\rho_{\text{ideal}}, \rho_{\text{mit}})$	-	-	-	-	0.974	0.962	0.947	0.913	0.784
$F(\rho_{\text{ideal}}, \rho_{\text{exp}})$	0.988	0.986	0.977	0.967	0.96	0.943	0.923	0.877	0.714

Table 2: Fidelity of the prepared Werner states to the ideal states. For $p \geq 0.6$ readout error correction is applied (ρ_{mit}). Circuits run on 02.08.2022 with ibmq_manila.

We might wonder why the Bell state prepared in section 4.7 had a fidelity of 0.982 to the ideal state, while here the Werner state for $p = 1$, which is the same state and also run on the same device, has much less. The 2 CNOT gates used to prepare $G|00\rangle$, which have gate error rates of around 1%, cannot have such a strong effect. The potential cause could be the implementation of the mid circuit measurements with high the readout errors or long execution time. It would be an interesting future project to pinpoint the important factors behind this low fidelity.

6 Simulation of unitary dynamics

In this section we will look at how to simulate unitary dynamics on a quantum computer. Even if we have a finite unitary matrix describing the process, there is no general way to simulate this on a quantum computer. The problem here is finding a circuit of 2-qubit gates which approximates this dynamics and has a circuit depth small enough, in order for the the implementation time of the circuit to be not longer than the decoherence time.

In the following sections we will show ways to solve this problem in the case where the unitary is a physical Hamiltonian. First we will look at the transverse Ising model as an example of simulating unitary dynamics for exactly solvable models [37] [38], then have a short outlook at the trotter method which can be used for approximately simulating arbitrary unitary dynamics.

6.1 Exactly solvable system - Transverse Ising Hamiltonian

We first look at the analytical solution for the antiferromagnetic Ising spin chain [39, 40] and then use this solution to simulate the system on a quantum computer. The Hamiltonian of the system is

$$H = \sum_{i=1}^{n} \sigma_i^x \sigma_{i+1}^x + \lambda \sum_{i=1}^{n} \sigma_i^z \,. \tag{40}$$

This describes a circular spin chain. The first term describes the nearest neighbor interaction. If the Hamiltonian consisted only of this term the spins would favor being aligned in the x-direction and in the opposite direction of the neighboured spins in the chain. But the σ^z part, which describes a transverse field in the z-direction effecting all spins, makes the solution for the Hamiltonian not trivial anymore.

Jordan-Wigner Transformation

Let us now proceed to transform the Hamiltonian (40) into a diagonal form. First we define annihilation and creation operators using the pauli operators by

$$a_j^+ = \left(\sigma_j^x + i\sigma_j^y\right)/2 \quad \text{and} \quad a_j = \left(\sigma_j^x - i\sigma_j^y\right)/2 \tag{41}$$

Then we get

$$\sigma_i^x = a_i^+ + a_i \qquad \sigma_i^y = -i\left(a_i^+ + a_i\right) \qquad \sigma_i^z = 2a_i^+ a_i - 1 \tag{42}$$

and the Hamiltonian in terms of a_i^+ and a_i is

$$H = \sum_{i=1}^{n} \left(a_i^+ a_{i+1}^+ + a_i^+ a_{i+1} + a_i a_{i+1}^+ + a_i a_{i+1}\right) + 2\lambda \sum_{i=1}^{n} a_i^+ a_i - n\lambda \tag{43}$$

The constant term $-n\lambda$ can be omitted. The annihilation and creation operators fulfill the fermionic anticommutation relation for *same sites*

$$\{a_i, a_i^+\} = 1 \quad \text{and} \quad a_i^2 = 0, \tag{44}$$

but $a_i^{(+)}$ and $a_j^{(+)}$ commute for $i \neq j$. We can get fully fermionic commutation relations by performing the Jordan-Wigner Transformation [41]. The Jordan-Wigner Transformation is given by

$$c_j = exp\left(\pi i \sum_{l=1}^{j-1} a_l^+ a_l\right) a_j \quad \text{and} \quad c_j^+ = a_j^+ \exp\left(-\pi i \sum_{l=1}^{j-1} a_l^+ a_l\right) \tag{45}$$

The inverse Transformation is

$$a_j = \exp\left(-\pi i \sum_{l=1}^{j-1} c_l^+ c_l\right) c_j \quad \text{and} \quad a_j^+ = c_j^+ \exp\left(\pi i \sum_{l=1}^{j-1} c_l^+ c_l\right) \tag{46}$$

Now we get for $i \leq j$

$$\begin{aligned}
c_i c_j^+ &= \exp\left(\pi i \sum_{l=1}^{i-1} a_l^+ a_l\right) a_i a_j^+ \exp\left(-\pi i \sum_{l=1}^{j-1} a_l^+ a_l\right) \\
&= \exp\left(\pi i \sum_{l=1}^{i-1} a_l^+ a_l\right) a_j^+ a_i \exp\left(-\pi i \sum_{l=1}^{j-1} a_l^+ a_l\right) \\
&= a_j^+ \exp\left(\pi i \sum_{l=1}^{i-1} a_l^+ a_l\right) \exp\left(-\pi i \left[\sum_{l=1}^{j-1} a_l^+ a_l - 1\right]\right) a_i \\
&= c_j^+ \exp(\pi i) c_i = -c_j^+ c_i
\end{aligned}$$

Same for $j \leq i$ and $c_j c_j^+ = a_j a_j^+$ as well as $c_j^+ c_j = a_j^+ a_j$, so we indeed have the fermionic commutation rule also for *different sites*

$$\{c_i, c_j^+\} = \delta_{ij} \tag{47}$$

Let us now look at the transformation of the terms in the Hamiltonian.

$$\begin{aligned}
a_j^+ a_{j+1} &= c_j^+ \exp\left(\pi i \sum_{l=1}^{j-1} c_l^+ c_l\right) \exp\left(-\pi i \sum_{l=1}^{j} c_l^+ c_l\right) c_{j+1} \\
&= c_j^+ \exp(-\pi c_j^+ c_j) c_{j+1} = c_j^+ c_{j+1}
\end{aligned} \tag{48}$$

$$\begin{aligned}
a_j^+ a_{j+1}^+ &= c_j^+ \exp\left(\pi i \sum_{l=1}^{j-1} c_l^+ c_l\right) c_{j+1}^+ \exp\left(\pi i \sum_{l=1}^{j} c_l^+ c_l\right) \\
&= c_j^+ \exp(\pi c_j^+ c_j) c_{j+1}^+ = c_j^+ c_{j+1}^+
\end{aligned} \tag{49}$$

So the Hamiltonian in the fermionic operators reads:

$$H = \sum_{i=1}^{n} \left(c_i^+ c_{i+1} + c_{i+1}^+ c_i + c_i c_{i+1} + c_i^+ c_{i+1}^+\right) + 2\lambda \sum_{i=1}^{n} c_i^+ c_i \tag{50}$$

Fourier Transform

Now we do a (quantum) Fourier transformation to transform the Hamiltonian to momentum space. One can easily check that that fermionic commutation relations are preserved by a Fourier transformation. The Fourier transformation is given by

$$b_k^+ = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} e^{\frac{2\pi ikj}{n}} c_j^+ \quad \text{and} \quad b_k = \frac{1}{\sqrt{n}} \sum_{j=1}^{n} e^{-\frac{2\pi ikj}{n}} c_j \tag{51}$$

with the inverse transformation

$$c_j^+ = \frac{1}{\sqrt{n}} \sum_{k=1}^{n} e^{-\frac{2\pi ikj}{n}} b_k^+ \quad \text{and} \quad c_j = \frac{1}{\sqrt{n}} \sum_{k=1}^{n} e^{\frac{2\pi ikj}{n}} b_k \tag{52}$$

We then obtain

$$\begin{aligned} \sum_{j=1}^{n} c_j^+ c_{j+a} &= \frac{1}{n} \sum_{j=1}^{n} \sum_{k,l=1}^{n} \exp\left(\frac{2\pi i}{n}(-jk + (j+a)l)\right) b_k^+ b_l \\ &= \frac{1}{n} \sum_{j=1}^{n} \exp\left(\frac{2\pi i}{n} j(l-k)\right) \sum_{k,l=1}^{n} \exp\left(\frac{2\pi i}{n} la\right) b_k^+ b_l \\ &= \delta_{lk} \sum_{k,l=1}^{n} \exp\left(\frac{2\pi i}{n} la\right) b_k^+ b_l = \sum_{k=1}^{n} \exp\left(\frac{2\pi i}{n} ka\right) b_k^+ b_k \end{aligned} \tag{53}$$

and similarly

$$\sum_{j=1}^{n} c_j c_{j+1} = \sum_{k=1}^{n} \exp\left(\frac{2\pi i}{n} k\right) b_{-k} b_k \tag{54}$$

Then

$$\begin{aligned} \sum_{j=1}^{n} c_j^+ c_{j+1} + c_{j+1}^+ c_j + 2\lambda c_j^+ c_j &= \sum_{k=1}^{n} \left[\exp\left(\frac{2\pi i}{n} k\right) + \exp\left(-\frac{2\pi i}{n} k\right) + 2\lambda\right] b_k^+ b_k \\ &= \sum_{k=1}^{n} 2\left[\cos\left(\frac{2\pi k}{n}\right) + \lambda\right] b_k^+ b_k \end{aligned} \tag{55}$$

Now we use the symmetry $k \to -k$ and the anti-commutation:

$$\begin{aligned}
\sum_{j=1}^{n} c_j c_{j+1} + c_{j+1}^+ c_j^+ &= \sum_{k=1}^{n} \left[\exp\left(\frac{2\pi i}{n} k\right) b_{-k} b_k + \exp\left(-\frac{2\pi i}{n} k\right) b_k^+ b_{-k}^+ \right] \\
&= \frac{1}{2} \sum_{k=1}^{n} \left[\exp\left(\frac{2\pi i}{n} k\right) b_{-k} b_k + \exp\left(-\frac{2\pi i}{n} k\right) b_k^+ b_{-k}^+ \right] \\
&\quad + \frac{1}{2} \sum_{k=1}^{n} \left[\exp\left(-\frac{2\pi i}{n} k\right) b_k b_{-k} + \exp\left(\frac{2\pi i}{n} k\right) b_{-k}^+ b_k^+ \right] \\
&= \frac{1}{2} \sum_{k=1}^{n} \left[\exp\left(\frac{2\pi i}{n} k\right) - \exp\left(-\frac{2\pi i}{n} k\right) \right] \left(b_{-k} b_k + b_{-k}^+ b_k^+ \right) \\
&= i \sum_{k=1}^{n} \sin\left(\frac{2\pi k}{n}\right) \left(b_{-k} b_k + b_{-k}^+ b_k^+ \right) \qquad (56)
\end{aligned}$$

So the Hamiltonian in the Fourier transformed variables is

$$H = \sum_{k=1}^{n} 2 \left[\cos\left(\frac{2\pi k}{n}\right) + \lambda \right] b_k^+ b_k + i \sin\left(\frac{2\pi k}{n}\right) \left(b_{-k} b_k + b_{-k}^+ b_k^+ \right) \qquad (57)$$

Bogoliubov transformation

The Hamiltonian in momentum space (eq. 57) is almost diagonal, but there still are the $b_{-k} b_k - h.c.$ terms left. These can be separated by applying a Bogoliubov transformation, which in general is given by

$$a_k = \vec{U}_k \begin{pmatrix} b_k \\ b_{-k} \end{pmatrix} + \vec{V}_k \begin{pmatrix} b_k^+ \\ b_{-k}^+ \end{pmatrix} \quad \text{and} \quad a_k^+ = \vec{U}_k^{\,*} \begin{pmatrix} b_k^+ \\ b_{-k}^+ \end{pmatrix} + \vec{V}_k^{\,*} \begin{pmatrix} b_k \\ b_{-k} \end{pmatrix} \qquad (58)$$

The fermionic anticommutation relations give these conditions for the coefficients:

$$\{a_k, a_k\} = \vec{U}_k \vec{V}_k \stackrel{!}{=} 0 \quad \text{and} \quad \{a_k, a_k^+\} = |\vec{U}_k|^2 + |\vec{V}_k|^2 \stackrel{!}{=} 1 \qquad (59)$$

So we choose

$$\vec{U}_k = \begin{pmatrix} e^{i\phi_k} u_k \\ 0 \end{pmatrix} = \begin{pmatrix} e^{i\phi_k} \cos\left(\frac{\Theta_k}{2}\right) \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{V}_k = \begin{pmatrix} 0 \\ e^{i\Phi_k} v_k \end{pmatrix} = \begin{pmatrix} 0 \\ e^{i\Phi_k} \sin\left(\frac{\Theta_k}{2}\right) \end{pmatrix} \qquad (60)$$

As the diagonal Hamiltonian is of the form $\sum_k \omega_k a_k^+ a_k$ with ω_k real, comparing the complex phases in

$$a_k^+ a_k = u_k^2 b_k^+ b_k + v_k^2 b_{-k} b_{-k}^+ + u_k v_k \left(e^{i(\phi_k - \Phi_k)} b_{-k} b_k - e^{i(\Phi_k - \phi_k)} b_{-k}^+ b_k^+ \right) \qquad (61)$$

with eq. (57) we see that

$$e^{i(\phi_k - \Phi_k)} = i \quad \text{and} \quad e^{i(\Phi_k - \phi_k)} = -i \qquad (62)$$

We choose $\phi_k = 0$ and $\Phi_k = -\frac{\pi}{2}$, which leads to the transformation

$$a_k = u_k b_k - i v_k b_{-k}^+ \quad \text{and} \quad a_k^+ = u_k b_k^+ + i v_k b_{-k} \tag{63}$$

with $u_k = \cos\left(\frac{\Theta_k}{2}\right)$ and $v_k = \sin\left(\frac{\Theta_k}{2}\right)$.

To calculate Θ_k and ω_k we start with the diagonal Hamiltonian

$$\begin{aligned} H = \sum_k \omega_k a_k^+ a_k &\overset{(61)}{=} \sum_k \omega_k \left[u_k^2 b_k^+ b_k + v_k^2 b_{-k} b_{-k}^+ + i u_k v_k \left(b_{-k} b_k + b_{-k}^+ b_k^+\right)\right] \\ &= \sum_k \omega_k \left[u_k^2 b_k^+ b_k + v_k^2 (1 - b_{-k}^+ b_{-k}) + i u_k v_k \left(b_{-k} b_k + b_{-k}^+ b_k^+\right)\right] \\ &= \sum_k \omega_k \left[(u_k^2 - v_k^2) b_k^+ b_k + v_k^2 + i u_k v_k \left(b_{-k} b_k + b_{-k}^+ b_k^+\right)\right] \\ &\equiv \sum_k \omega_k \left[\cos(\Theta_k) b_k^+ b_k + \frac{i}{2} \sin(\Theta_k) \left(b_{-k} b_k + b_{-k}^+ b_k^+\right)\right] \end{aligned} \tag{64}$$

and ignore the constant term $\sum_k \omega_k v_k^2$. In line 3 we use that we sum over all possible values for k and thus can make the index shift $k \to -k$ for the second term of line 2. The assumptions $\omega_k = \omega_{-k}$ and $\Theta_k = \Theta_{-k}$ (so $v_k = v_{-k}$) can be verified later (eq. 68 and 69). Comparing with eq. (57) gives

$$\omega_k \cos(\Theta_k) = 2\left[\cos\left(\frac{2\pi k}{n}\right) + \lambda\right] \tag{65}$$

$$\frac{\omega_k}{2} \sin(\Theta_k) = \sin\left(\frac{2\pi k}{n}\right) \tag{66}$$

Therefore:

$$\left(\frac{\omega_k}{2}\right)^2 = \left(\frac{\omega_k}{2}\right)^2 \left(\cos(\Theta_k)^2 + \sin(\Theta_k)^2\right) = \left(\cos\left(\frac{2\pi k}{n}\right) + \lambda\right)^2 + \sin\left(\frac{2\pi k}{n}\right)^2 \tag{67}$$

For $k \neq \frac{n}{2}$ we have $\omega_k^2(\lambda) > 0$, so we have to choose either $\omega_k(\lambda) > 0$ or $\omega_k(\lambda) < 0$ to have a continuous solution for $\omega_k(\lambda)$. We take $\omega_k(\lambda) > 0$ (then excited states of the diagonal Hamiltonian correspond to excited qubit states), so

$$\omega_k = 2\sqrt{\left(\cos\left(\frac{2\pi k}{n}\right) + \lambda\right)^2 + \sin\left(\frac{2\pi k}{n}\right)^2} \tag{68}$$

$$\Theta_k \overset{(65)}{=} \arccos\left[\frac{\cos\left(\frac{2\pi k}{n}\right) + \lambda}{\sqrt{\left(\cos\left(\frac{2\pi k}{n}\right) + \lambda\right)^2 + \sin\left(\frac{2\pi k}{n}\right)^2}}\right] \tag{69}$$

for $k \neq \frac{n}{2}$. For $k = \frac{n}{2}$ we get for any λ

$$\omega_{\frac{n}{2}} \sin(\Theta_{\frac{n}{2}}) \overset{(66)}{=} 0 \tag{70}$$

By choosing $\Theta_{\frac{n}{2}} = 0$ eq. (65) becomes

$$\omega_{\frac{n}{2}} = 2(\lambda - 1) \tag{71}$$

6.2 Simulation of the Ising model on a quantum computer

We now look at each step of the analytical solution again and express it as a unitary transformations, which then can be represented with quantum gates. These circuits, added one after another then give the circuit U_{dis}. The inverse of this circuit then allows to prepare eigenstates of the Hamiltonian by starting in it's diagonal form.

Quantum circuit for the Jordan-Wigner Transformation

The first transformation from just changes the variables from the Pauli matrices to the annihilation and creation operators $a_i^{(+)}$. Thus we do not need to implement anything here.

As the Jordan-Wigner transformation maps the spin-1/2 states $|i_1 \dots i_n\rangle$ to fermionic states $(c_1^+)^{i_1} \dots (c_n^+)^{i_n} |\Omega_c\rangle$, where $|\Omega_c\rangle$ is the vaccumn of the fermions, the coefficients $\Psi_{i_1 \dots i_n}$ of the wave function do not change:

$$|\Psi\rangle = \sum_{i_1,\dots,i_n} \Psi_{i_1 \dots i_n} |i_1 \dots i_n\rangle = \sum_{i_1,\dots,i_n} \Psi_{i_1 \dots i_n} (c_1^+)^{i_1} \dots (c_n^+)^{i_n} |\Omega_c\rangle . \tag{72}$$

This means that we do not need to do anything on the quantum computer to implement this transformation, the only change is that the qubits now represent fermions.

It is however important to keep in mind that, when one swaps two qubits one should add a minus sign if the qubits are in state $|11\rangle$ (corresponding to two occupied fermionic states) to simulate the fermionic anticommutation. So if we have swap gates in our circuit, controlled-Z gates need to be added after them.[31]

Quantum circuit for the Fourier Transformation

We now represent the Fourier Transformation with two-qubit unitary matrices, which can then easily be represented with quantum gates. First we look at the simplest case of just $n = 2$ qubits, so the Fourier transform is given by

$$b_k^+ = \frac{1}{\sqrt{n}} \sum_{j=0}^{n-1} e^{\frac{2\pi i k j}{n}} c_j^+ = \frac{1}{\sqrt{2}} \left(c_0^+ + e^{i\pi k} c_1^+ \right) \tag{73}$$

with the inverse transformation

$$c_j^+ = \frac{1}{\sqrt{n}} \sum_{k=0}^{n-1} e^{-\frac{2\pi i k j}{n}} b_k^+ = \frac{1}{\sqrt{2}} \left(b_0^+ + e^{i\pi j} b_1^+ \right) \tag{74}$$

We can find the unitary for this transformation by looking at the effect it has on the 4 pure states $|00\rangle, |01\rangle, |10\rangle, |11\rangle$ (be aware that Qiskit flips the order of the tensor

[31] SWAP gates inserted by the Qiskit transpiler do not matter here, as they go along with a permutation of the mapping of virtual qubits to physical qubits.

product, so $|01\rangle \to |10\rangle$ etc., **todo: mention in sections before**):

$$\begin{aligned}
|01\rangle &= c_1^+ |00\rangle = \frac{1}{\sqrt{2}}\left(b_0^+ - b_1^+\right)|00\rangle = \frac{1}{\sqrt{2}}\left(|10\rangle - |01\rangle\right) \\
|10\rangle &= c_0^+ |00\rangle = \frac{1}{\sqrt{2}}\left(b_0^+ + b_1^+\right)|00\rangle = \frac{1}{\sqrt{2}}\left(|10\rangle + |01\rangle\right) \\
|11\rangle &= c_0^+ c_1^+ |00\rangle = \frac{1}{2}\left(b_0^+ + b_1^+\right)\left(b_0^+ - b_1^+\right)|00\rangle = \frac{1}{2}\left({b_0^+}^2 - b_0^+ b_1^+ + b_1^+ b_0^+ - {b_1^+}^2\right)|00\rangle \\
&= -b_0^+ b_1^+ |00\rangle = -|11\rangle
\end{aligned}$$

So the unitary for $n = 2$ is

$$F_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = F_0^+ \tag{75}$$

For convenience we will flip this gate, so further use

$$F_0 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & -1 \end{pmatrix} = F_0^+ \tag{76}$$

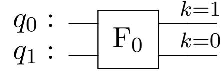

Figure 19: The F_0 gate, the output on q_0 corresponds to $k = 1$ and on q_1 to $k = 0$.

For $n \geq 4$ we implement the fast Fourier transform as a quantum circuit, which we will first do for $n = 4$ and then interactively continue for $n_{new} = 2n$. For $n = 4$ the transformation is given by

$$b_k^+ = \frac{1}{\sqrt{4}} \sum_{j=0}^{3} e^{\frac{2i\pi kj}{4}} b_k^+ = \left[c_0^+ + e^{i\pi k} c_2^+\right] + e^{i\pi k/2}\left[c_1^+ + e^{i\pi k} c_3^+\right] \tag{77}$$

The first and second term looks like eq. (73), so we first we 'split' the qubits into even and odd numbered ones and then apply the gate F_0 between those (see Figure 20). For k even and odd we get:

$$b_{2k}^+ = \left[c_0^+ + c_2^+\right] + e^{i\pi k}\left[c_1^+ + c_3^+\right] \tag{78}$$

$$b_{2k+1}^+ = \left[c_0^+ - c_2^+\right] + e^{i\pi 1/2} e^{i\pi k}\left[c_1^+ - c_3^+\right] \tag{79}$$

Eq. (78) is the same transformation like we had for $n = 2$, just taking the qubits which correspond to $k = 0$ (q_2 and q_3) as input. For odd k we have the qubits corresponding

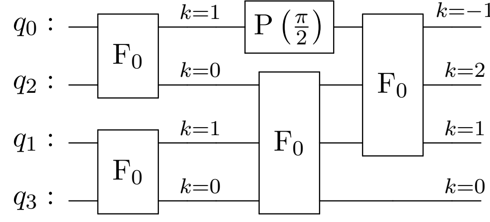

Figure 20: The circuit for $n = 4$. Each gate here is a two qubit gate.

the $k = 1$ (q_0 and q_1) as input and the gate F_0 preceded by a phase shift of $\pi/2$ on the second - as we flipped the gate the first - qubit.

Now we assume we have a circuit for n qubits and want to get the circuit for $m = 2n$ qubits. The transformation is given by

$$b_k^+ = \sum_{j=0}^{m-1} e^{\frac{2\pi ik}{m}j} c_j^+ = \sum_{j'=0}^{\frac{m}{2}-1} \left[e^{\frac{2\pi ik}{m}2j'} c_{2j'}^+ + e^{\frac{2\pi ik}{m}(2j'+1)} c_{2j'+1}^+ \right]$$

$$= \sum_{j'=0}^{n-1} e^{\frac{2\pi ik}{n}j'} c_{2j'}^+ + e^{\frac{2\pi ik}{m}} \sum_{j'=0}^{n-1} e^{\frac{2\pi ik}{n}j'} c_{2j'+1}^+ \quad (80)$$

$$\text{For } k = 0, \ldots, n-1: \quad b_{-n+k}^+ = b_{k+n}^+ = \sum_{j'=0}^{n-1} e^{\frac{2\pi ik}{n}j'} c_{2j'}^+ - e^{\frac{2\pi ik}{m}} \sum_{j'=0}^{n-1} e^{\frac{2\pi ik}{n}j'} c_{2j'+1}^+ \quad (81)$$

So we first apply the circuits we have found for n sites on the two sets of even and odd numbered qubits. Then we take each two qubits corresponding to $k = 0, \ldots, n-1$ and apply to them this F_k^m gate:

q0 : P $\left(\frac{2\pi k}{m}\right)$ F0 k=1
q1 : k=0

$$F_k^m = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{e^{\frac{2\pi ik}{m}}}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{e^{\frac{2\pi ik}{m}}}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & -e^{\frac{2\pi ik}{m}} \end{pmatrix} \quad (82)$$

Figure 21: F_k^m gate, an F_0 gate which on its first qubit is preceded by a $\frac{2\pi k}{m}$ phase gate.

For $n = 8$ this gives the circuit shown in Figure 22 (using that the F_k^n gate is equal to F_{2k}^m gate and just writing F_k instead of F_k^m).

The last step to implement the Fourier Transform as a quantum circuit is now to find a representation for F_0 by quantum gates known to Qiskit, which then decomposes these gates into the basis gates for the quantum machine we run the circuit on.

In [37] the a circuit for F_0 is given (see Figure 23), but this uses 4 CNOT gates and Qiskit's unitary synthesis (see section 4.8) can generate the circuit to implement F_0 with

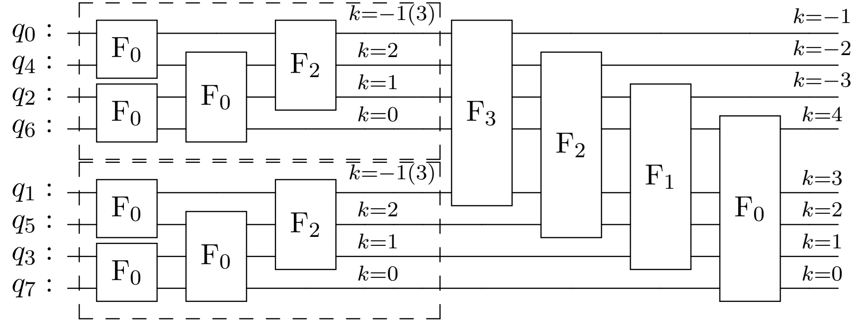

Figure 22: The circuit for $n = 8$ using the circuit for $n = 4$ twice. Each gate here is a two qubit gate.

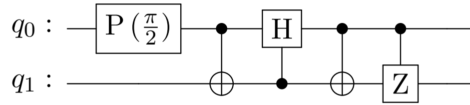

Figure 23: Circuit to implement the gate F_0 given in [37]. The controlled Hadamard gate and controlled Z gates are each realized using 1 CNOT gate and single qubit gates. So in total we have 4 CNOT gates.

only 2 CNOTs. So we use the circuit generated by Qiskit instead. This gives the following decomposition of F_0 into CNOT gates and the single qubit gate U_3:.

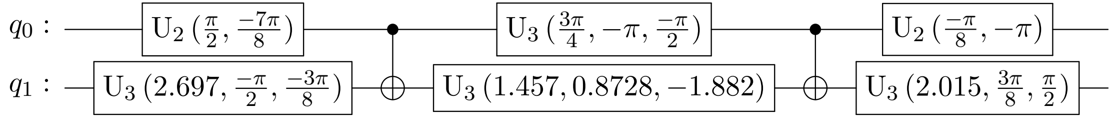

Figure 24: Circuit to implement the gate F_0 using only 2 CNOT gates, generated by Qiskit. The U_3 gate implements a general single qubit rotation.

Quantum circuit for the Bogoliubov transformation

Now we want to express the Bogoliubov transformation transformation as a quantum circuit. From eq. 63 we have

$$a_k = u_k b_k - i v_k b_{-k}^{+} \quad \text{and} \quad a_k^{+} = u_k b_k^{+} + i v_k b_{-k} \tag{83}$$

with $u_k = \cos\left(\frac{\Theta_k}{2}\right)$ and $v_k = \sin\left(\frac{\Theta_k}{2}\right)$.

Looking at the Transformation we see that it can be split into $\frac{n}{2}$ parallel transformations, each affecting only two qubits corresponding to momentum $-k$ and k.

First we express the Bogoliubov transformation as a two-qubit unitary B_k^n by looking at the pure two-qubit states. We write for convenience $|a\rangle_{-k}|b\rangle_k \equiv |ab\rangle$.

To find the inverse transformation of $|01\rangle$ we look at the inverse transformations of the number operators $n_{\pm k} = a_{\pm k}^+ a_{\pm k}$ applied on the state $|01\rangle$:

$$a_{-k}^+ a_{-k}|01\rangle = 0 \qquad \xrightarrow{B_k^{n+}} \left[u_k^2 b_{-k}^+ b_{-k} + v_k^2 b_k b_k^+ + i u_k v_k \left(b_k b_{-k} + b_k^+ b_{-k}^+\right)\right]|\Psi\rangle = 0 \tag{84}$$

$$a_k^+ a_k|01\rangle = |01\rangle \quad \xrightarrow{B_k^{n+}} \left[u_k^2 b_k^+ b_k + v_k^2 b_{-k} b_{-k}^+ + i u_k v_k \left(b_{-k} b_k + b_{-k}^+ b_k^+\right)\right]|\Psi\rangle = |\Psi\rangle \tag{85}$$

where $|\Psi\rangle = B_k^{n+}|01\rangle$. As $|\Psi\rangle = |01\rangle$ fulfills both equations, we have $B_k^{n+}|01\rangle = |01\rangle$. Analogously for $|01\rangle$ we get $B_k^{n+}|10\rangle = |10\rangle$. Then we can use that these two states are invariant under the transformation for finding the transformation of the other two pure states:

$$|00\rangle = a_k|01\rangle \qquad \xrightarrow{B_k^{n+}} \qquad \left[u_k b_k - i v_k b_{-k}^+\right]|01\rangle = u_k|00\rangle - i v_k|11\rangle \tag{86}$$

$$|11\rangle = -a_k^+|10\rangle \qquad \xrightarrow{B_k^{n+}} \qquad -\left[u_k b_k^+ + i v_k b_{-k}\right]|10\rangle = u_k|11\rangle - i v_k|00\rangle \tag{87}$$

The minus sign in the last equation is due to $|11\rangle = a_{-k}^+ a_k^+|00\rangle = -a_k^+ a_{-k}^+|00\rangle$.
So the unitary for the (inverse) Bogoliubov transformation at the qubits corresponding to momentum $-k$ and k is

$$B_k^{n+} = \begin{pmatrix} u_k & 0 & 0 & -iv_k \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -iv_k & 0 & 0 & u_k \end{pmatrix} \quad \Longrightarrow \quad B_k^n = \begin{pmatrix} u_k & 0 & 0 & iv_k \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ iv_k & 0 & 0 & u_k \end{pmatrix}$$

with $u_k = \cos\left(\frac{\Theta_k}{2}\right)$, $v_k = \sin\left(\frac{\Theta_k}{2}\right)$ and Θ_k from eq. 69.
As $\Theta_0 = \Theta_{\frac{n}{2}} = 0$ we have $B_0^n = \mathbb{I}$ and thus no gate needed here.

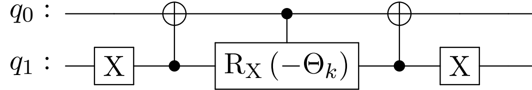

Figure 25: Circuit to implement the gate B_k^n for $0 < k < \frac{n}{2}$ given in [37]. The R_X gate is implemented using two CNOT gates and single qubit gates, so in total we have 4 CNOT gates.

For $0 < k < \frac{n}{2}$ we now look to express the unitary B_k^n in a quantum circuit. In [37] the a circuit for B_k^n is given, but this uses 4 CNOT gates and Qiskit's unitary synthesis (see section 4.8) can generate the circuit to implement B_k^n with 3 or less CNOT gates (depending on the exact value for Θ_k). So we use the circuit generated by Qiskit instead.

The full circuit for the diagonalisation of the $n = 4$ spin chain is shown in Figure 26. Analogously for $n = 8$ where take the circuit from Figure 22 and attach the gates B_k to the qubits corresponding to momentum $\pm k$ for $k = 1, 2, 3$

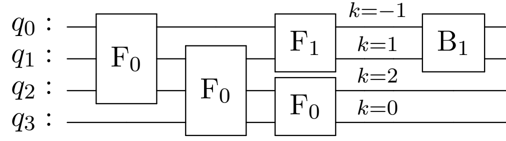

Figure 26: The circuit for the diagonalisation of the $n = 4$ spin chain. Each gate here is a two qubit gate.

Preparing the ground state of the Ising spin chain

We have the Hamiltonian in diagonal form $H = \sum_k \omega_k a_k^+ a_k$ with

$$\omega_k = 2 \begin{cases} \sqrt{\left(\cos\left(\frac{2\pi k}{n}\right) + \lambda\right)^2 + \sin\left(\frac{2\pi k}{n}\right)^2} & \text{for } 0 \leq k < \frac{n}{2} \\ \lambda - 1 & \text{for } k = \frac{n}{2} \end{cases} \tag{88}$$

and a quantum circuit to diagonalize the Hamiltonian.
For $n = 4$ the ground state of H is

$$|gs\rangle = \begin{cases} |0010\rangle & \text{for } \lambda \leq 1 \\ |0000\rangle & \text{for } \lambda > 1 \end{cases} \tag{89}$$

as q_2 corresponds to $k = \frac{n}{2}$.
Having prepared this state on a quantum machine one can now apply the inverse of the circuit from Figure 26 to prepare the ground state of the $n = 4$ spin chain.

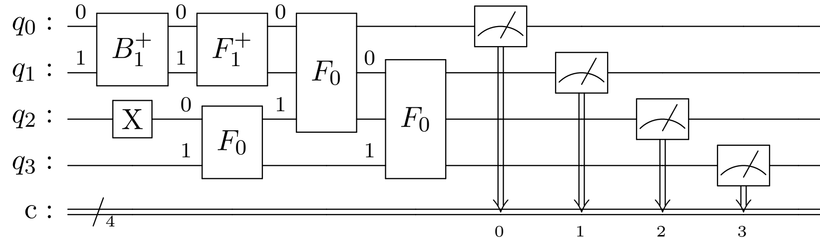

Figure 27: The circuit for the preparing the ground state of the $n = 4$ spin chain for $\lambda < 1$. For $\lambda > 1$ the Pauli X gate is removed from the circuit. Each gate here is a two qubit gate.

This can then be used to e.q. measure the ground state magnetization by doing a standard (Pauli Z basis) measurement on every qubit and then calculating the sum of the expectation values for all measurements.

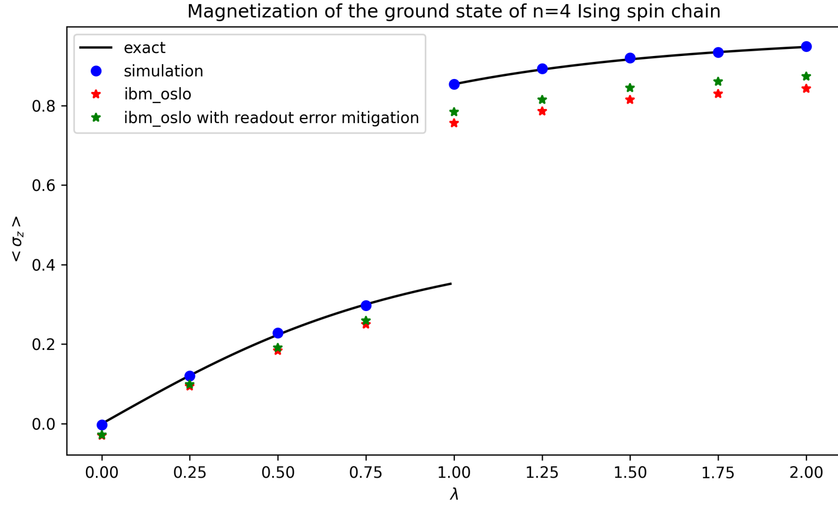

Figure 28: Magnetization of the $n = 4$ Ising spin chain ground state run on IBM Oslo on 02.08.2022.

Preparing a thermal state of the Ising spin chain

Now we want to prepare the state of the Ising spin chain at a arbitrary temperature T. The thermal state ensemble is given by the Boltzmann distribution:

$$\rho(\beta) = \frac{e^{-\beta H}}{\mathcal{Z}} = \frac{1}{\mathcal{Z}} \sum_i e^{-\beta \epsilon_i} |E_i\rangle \langle E_i| \text{ with } \mathcal{Z} = \sum_i e^{-\beta \epsilon_i} \tag{90}$$

We set $\hbar = 1$ and $k_B = 1$, so $\beta = \frac{1}{T}$. $|E_i\rangle$ are the eigenstates of H and ϵ_i the corresponding energies:

$$\epsilon_i = \langle E_i| H |E_i\rangle = \langle E_i| \sum_k \omega_k a_k^+ a_k |E_i\rangle \tag{91}$$

Here we have to keep in mind the assignment between k and the qubits of our circuits. For example the state for the four qubit circuit (see Figure 20) the mapping is

$$[q_0,\, q_1,\, q_2,\, q_3] \rightarrow [k = -1 \equiv 3,\, k = 1,\, k = 2,\, k = 0] \tag{92}$$

So for the eigenstate $|1100\rangle$ we get the energy

$$\epsilon = \langle 1100| \sum_k \omega_k a_k^+ a_k |1100\rangle = \omega_3 + \omega_1 = 2\sqrt{\lambda^2 + 1} + 2\sqrt{\lambda^2 + 1} = 4\sqrt{\lambda^2 + 1} \tag{93}$$

Now we want to prepare the ensemble of states from (90). Like in section 5.2 we first prepare the statevector

$$|I\rangle = \frac{1}{\sqrt{\mathcal{Z}}} \sum_i e^{-\frac{\beta}{2}\epsilon_i} |E_i\rangle = \frac{1}{\sqrt{\mathcal{Z}}} \sum_i \left(\prod_k e^{-\frac{\beta}{2}\omega_k a_k^+ a_k} \right) |E_i\rangle \tag{94}$$

and then measure all qubits to project the state into our desired state. As the Hamiltonian is diagonal, its eigenstates are all the Z basis states, thus we can write:

$$\{|E_i\rangle\}_i = \{|0\rangle, |1\rangle\}^n \text{ with } |E_i\rangle = \bigotimes_k |E_i\rangle_k \tag{95}$$

where we refer to the qubits by their assigned value of k. Then we get from (94)

$$|I\rangle = \frac{1}{\sqrt{\mathcal{Z}}} \sum_i \left(\bigotimes_k e^{-\frac{\beta}{2}\omega_k a_k^+ a_k} |E_i\rangle_k \right) = \frac{1}{\sqrt{\mathcal{Z}}} \bigotimes_k \left(|0\rangle_k + e^{-\frac{\beta}{2}\omega_k} |1\rangle_k \right) \tag{96}$$

As $\mathcal{Z}$ is just for normalization, this is equal to

$$|I\rangle = \bigotimes_k \frac{1}{\sqrt{\mathcal{Z}_k}} \left(|0\rangle_k + e^{-\frac{\beta}{2}\omega_k} |1\rangle_k \right) \tag{97}$$

with $Z_k = 1 + \exp(-\beta\omega_k/2)$.

As this is a product state we can easily prpare it on the quantum computer by a $R_y(\theta_k)$ rotation gate applied on each qubit, with

$$\theta_k = 2 \arccos\left(\frac{1}{\sqrt{\mathcal{Z}_k}}\right) \tag{98}$$

Now we look at the magnetization for thermal states with different values for β. As we expect for large β (corresponding to $T \to 0$ we get a similar result to the one for the ground state.

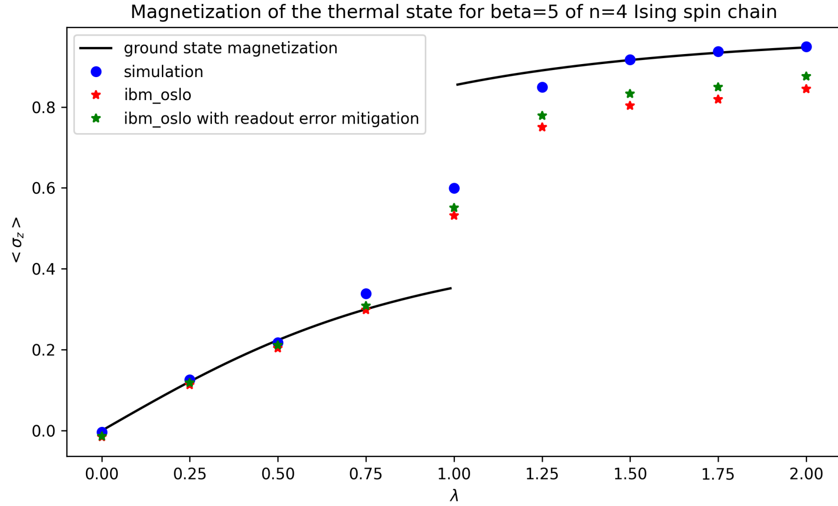

Figure 29: Magnetization of the $n = 4$ Ising spin chain thermal state with $\beta = 5$ run on IBM Oslo on 02.08.2022.

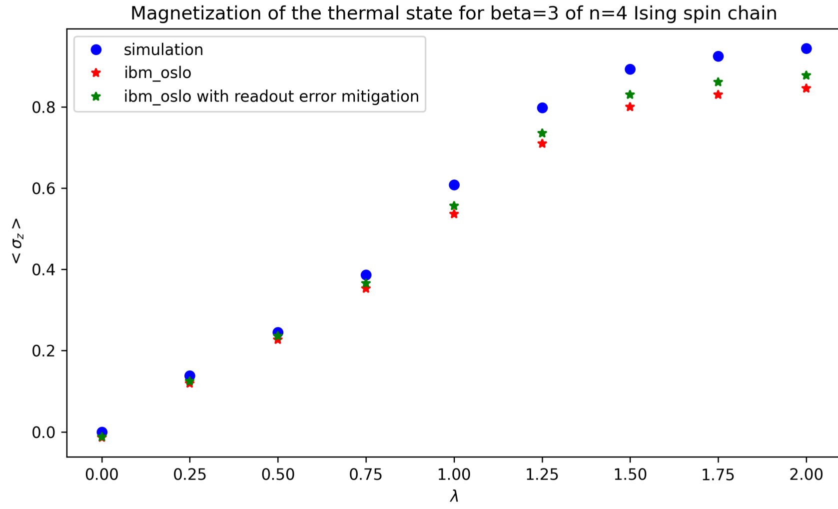

Figure 30: Magnetization of the $n = 4$ Ising spin chain thermal state with $\beta = 3$ run on IBM Oslo on 02.08.2022.

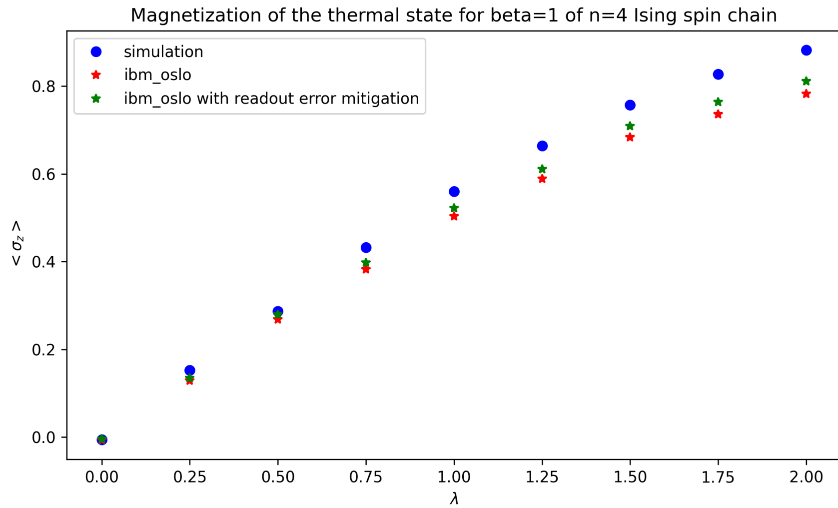

Figure 31: Magnetization of the $n = 4$ Ising spin chain thermal state with $\beta = 1$ run on IBM Oslo on 02.08.2022.

6.3 Trotter method

We now look at a method which allows us to simulate the unitary time evolution of a physical system independently of knowing an analytical solution for the system [42].

First the systems initial state is prepared by initiating the quantum computer ac-

cordingly. For example for a spin chain the state $|0\rangle$ corresponds to spin $\downarrow$ and $|1\rangle$ to $\uparrow$.

Consider as an example the time evolution of the spin system given by a Hamiltonian of the form

$$H = -J \sum_{i=1}^{n-1} \left(\sigma_i^x \sigma_{i+1}^x + \sigma_i^y \sigma_{i+1}^y \right) + U \sum_{i=1}^{n-1} \sigma_i^z \sigma_{i+1}^z + \sum_{i=1}^{n} \lambda_i \sigma_i^z \,. \tag{99}$$

So transform the initial state into the systems state after time t, we need to implement the unitary

$$U(t) = e^{-iHt} \tag{100}$$

by representing it by two-qubit gates.

As it is possible to decompose two-qubit unitaries into circuits with only a few CNOT gates, we would want to decompose U into terms like

$$e^{i\left[J\left(\sigma_i^x \sigma_{i+1}^x + \sigma_i^y \sigma_{i+1}^y\right) - U \sigma_i^z \sigma_{i+1}^z\right]t} \text{ and } e^{-i\lambda_i \sigma_i^z \Delta t} \,. \tag{101}$$

But if we desire a exact decomposition, then we have to consider the Zassenhaus formula:

$$e^{t(A+B)} = e^{tA} e^{tB} e^{-\frac{t^2}{2}[A,B]} e^{\frac{t^3}{6}([A,[A,B]]+2[B,[A,B]])} \dots \mathcal{O}(t^4) \tag{102}$$

The idea of the Suzuki-Trotter method is to implement the unitary $U(t)$ as a product of the unitaries for a small time step Δt:

$$U(t) = U(\Delta t)^M \quad \text{with } M = \frac{t}{\Delta t} \in \mathbb{N} \tag{103}$$

Then the commutator terms from (102) are of the order $(\Delta t)^2$ and thus can be neglected if Δt is small enough. So $U(\Delta t)$ is approximated by:

$$U(\Delta t) = e^{-iH\Delta t} = \prod_{i=1}^{n} e^{-i\lambda_i \sigma_i^z \Delta t} \prod_{i \text{ even}} e^{iH_i \Delta t} \prod_{i \text{ odd}} e^{iH_i \Delta t} + \mathcal{O}((\Delta t)^2) \tag{104}$$

with $H_i = J\left(\sigma_i^x \sigma_{i+1}^x + \sigma_i^y \sigma_{i+1}^y\right) - U \sigma_i^z \sigma_{i+1}^z$. $U(\Delta t)$ is then implemented by the quantum circuit in Figure 32. For approximating $U(t)$ this circuit is repeated M times.
Using the symmetric trotter formula from [43]

$$e^{t(A+B)} = e^{\frac{t}{2}A} e^{tB} e^{\frac{t}{2}A} + \mathcal{O}(t^3) \tag{105}$$

one can reduce the error of the approximation to the order of $(\Delta t)^3$:

$$U(\Delta t) = \prod_{i=1}^{n} e^{-i\lambda_i \sigma_i^z \frac{\Delta t}{2}} \prod_{i \text{ even}} e^{iH_i \frac{\Delta t}{2}} \prod_{i \text{ odd}} e^{iH_i \Delta t} \prod_{i \text{ even}} e^{iH_i \frac{\Delta t}{2}} \prod_{i=1}^{n} e^{-i\lambda_i \sigma_i^z \frac{\Delta t}{2}} + \mathcal{O}((\Delta t)^3) \tag{106}$$

It remains as a future project to investigate the quality of the Trotter method in a real implementation in comparison to the exact simulation obtained in the previous chapter.

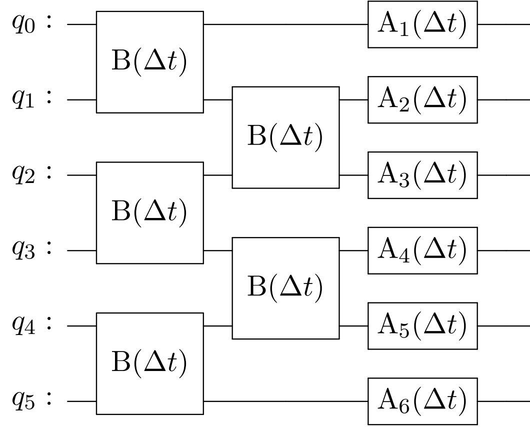

Figure 32: A Circuit to implement the trotter method for $n = 6$. The gates $B(\Delta t)$ implement the two-qubit unitary $\exp\left(iH_1\Delta t\right)$ with $H_1 = J\left(\sigma_1^x\sigma_2^x + \sigma_1^y\sigma_2^y\right) - U\sigma_1^z\sigma_2^z$ and the gates $A_i(\Delta t)$ the unitaries $\exp\left(-i\lambda_i\sigma_i^z\Delta t\right)$.

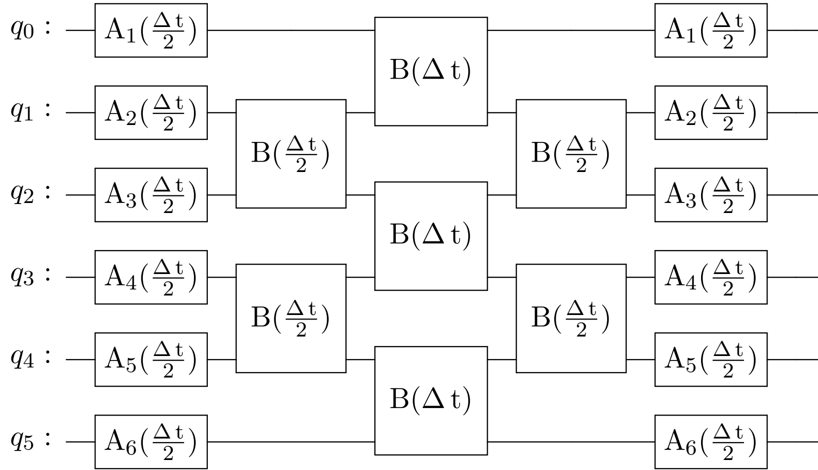

Figure 33: A Circuit to implement the symmetric trotterization for $n = 6$. The gates $B(\Delta t)$ implement the two-qubit unitary $\exp\left(iH_1\Delta t\right)$ with $H_1 = J\left(\sigma_1^x\sigma_2^x + \sigma_1^y\sigma_2^y\right) - U\sigma_1^z\sigma_2^z$ and the gates $A_i(\Delta t)$ the unitaries $\exp\left(-i\lambda_i\sigma_i^z\Delta t\right)$.

List of Tables

1 Execution times for *ibm_nairobi* as of June 29th 2022. The single qubit gate times are equal for all qubits, for the other times the average over all qubits is given. 6

2 Fidelity of the prepared Werner states to the ideal states. For $p \geq 0.6$ readout error correction is applied (ρ_{mit}). Circuits run on 02.08.2022 with ibmq_manila. 33

List of Figures

1 Circuit symbol and matrix for the CNOT gate. 3

2 Error map for the 7 qubit IBM device *ibm_nairobi*, showing readout and gate errors 7

3 Circuit for preparing $|\Psi^-\rangle$ 10

4 Circuit drawn using the `output='text'` (left) and `output='mpl'` (right) option. For a circuit drawn with the `output='latex_source'` option see Figure 3. . . 11

5 Circuit for preparing $|\Psi^-\rangle$, generated by *initialize* method. 12

6 Circuit for preparing $|\Psi^-\rangle$, generated by *initialize* method. Decomposed into standard gates. The $|0\rangle$ gate is a reset gate, which resets the qubit into the $|0\rangle$ state. 12

7 The Bloch sphere with the x, y and z axis marked. 14

8 Implementation of the measurement in X- and Y-basis. 15

9 On the IBM devices a swap gate is implemented using three CNOT gates. 19

10 Simple example for commuting gates optimization. 19

11 Example for converting a quantum circuit into a Directed Acyclic Graph (DAG). 20

12 Measurement results for prepared state $|\Psi^-\rangle$, without (blue) and with (red) readout error correction 27

13 The circuit used to prepare a $|\Psi^-\rangle$ state. 30

14 Preparing a mixed state 30

15 A circuit to prepare the Werner state on an IBM quantum computer using a conditional swap gate, which cannot be implemented currently. The conditional swap gate is the last gate of the circuit, it is only implemented if $c_3 = 1$, which is the case exactly if q_3 is measured in state $|1\rangle$. This circuit can be implemented once IBM allows for the use of gates conditionally implemented based on the state of the classical register, which is expected to be realized soon. 31

16 A circuit to simulate the Werner state on an IBM quantum computer. The gate $R(p)$ is given by $R_Y(2\arccos(\sqrt{p}))$. 31

17 Circuit for preparing Bell diagonal states. 32

18 Circuit for preparing the state from eq. (37) for $p = 0.6$ (so $p_{00} = p_{01} = p_{10} = 0.1$ and $p_{11} = 0.7$). 33

19 The F_0 gate, the output on q_0 corresponds to $k = 1$ and on q_1 to $k = 0$. . 40
20 The circuit for $n = 4$. Each gate here is a two qubit gate. 41
21 F_k^m gate, an F_0 gate which on its first qubit is preceded by a $\frac{2\pi k}{m}$ phase gate. 41
22 The circuit for $n = 8$ using the circuit for $n = 4$ twice. Each gate here is a two qubit gate. 42
23 Circuit to implement the gate F_0 given in [37]. The controlled Hadamard gate and controlled Z gates are each realized using 1 CNOT gate and single qubit gates. So in total we have 4 CNOT gates. 42
24 Circuit to implement the gate F_0 using only 2 CNOT gates, generated by Qiskit. The U_3 gate implements a general single qubit rotation. 42
25 Circuit to implement the gate B_k^n for $0 < k < \frac{n}{2}$ given in [37]. The R_X gate is implemented using two CNOT gates and single qubit gates, so in total we have 4 CNOT gates. 43
26 The circuit for the diagonalisation of the $n = 4$ spin chain. Each gate here is a two qubit gate. 44
27 The circuit for the preparing the ground state of the $n = 4$ spin chain for $\lambda < 1$. For $\lambda > 1$ the Pauli X gate is removed from the circuit. Each gate here is a two qubit gate. 44
28 Magnetization of the $n = 4$ Ising spin chain ground state run on IBM Oslo on 02.08.2022. 45
29 Magnetization of the $n = 4$ Ising spin chain thermal state with $\beta = 5$ run on IBM Oslo on 02.08.2022. 46
30 Magnetization of the $n = 4$ Ising spin chain thermal state with $\beta = 3$ run on IBM Oslo on 02.08.2022. 47
31 Magnetization of the $n = 4$ Ising spin chain thermal state with $\beta = 1$ run on IBM Oslo on 02.08.2022. 47
32 A Circuit to implement the trotter method for $n = 6$. The gates $B(\Delta t)$ implement the two-qubit unitary $\exp(iH_1\Delta t)$ with $H_1 = J(\sigma_1^x\sigma_2^x + \sigma_1^y\sigma_2^y) - U\sigma_1^z\sigma_2^z$ and the gates $A_i(\Delta t)$ the unitaries $\exp(-i\lambda_i\sigma_i^z\Delta t)$. 49
33 A Circuit to implement the symmetric trotterization for $n = 6$. The gates $B(\Delta t)$ implement the two-qubit unitary $\exp(iH_1\Delta t)$ with $H_1 = J(\sigma_1^x\sigma_2^x + \sigma_1^y\sigma_2^y) - U\sigma_1^z\sigma_2^z$ and the gates $A_i(\Delta t)$ the unitaries $\exp(-i\lambda_i\sigma_i^z\Delta t)$. 49

References

[1] D. P. DiVincenzo, "The physical implementation of quantum computation," *Fortschritte der Physik: Progress of Physics*, vol. 48, no. 9-11, pp. 771–783, 2000.

[2] J. Chow, O. Dial, and J. Gambetta, "Ibm quantum breaks the 100-qubit processor barrier," Nov 2021. [Online]. Available: https://research.ibm.com/blog/127-qubit-quantum-processor-eagle

[3] P. Meystre and M. Sargent, *Elements of quantum optics: With 218 problems.* Springer, 1999.

[4] A. Asfaw, T. Alexander, P. Nation, and J. Gambetta, "Get to the heart of real quantum hardware," Dec 2019. [Online]. Available: https://www.ibm.com/blogs/research/2019/12/qiskit-openpulse/

[5] A. Barenco, C. H. Bennett, R. Cleve, D. P. DiVincenzo, N. Margolus, P. Shor, T. Sleator, J. A. Smolin, and H. Weinfurter, "Elementary gates for quantum computation," *Physical review A*, vol. 52, no. 5, p. 3457, 1995.

[6] IBM, "Compute resources," 2022. [Online]. Available: https://quantum-computing.ibm.com/services?services=systems

[7] P. Schindler, D. Nigg, T. Monz, J. T. Barreiro, E. Martinez, S. X. Wang, S. Quint, M. F. Brandl, V. Nebendahl, C. F. Roos *et al.*, "A quantum information processor with trapped ions," *New Journal of Physics*, vol. 15, no. 12, p. 123012, 2013.

[8] D. S. Weiss and M. Saffman, "Quantum computing with neutral atoms," *Physics Today*, vol. 70, no. 7, pp. 44–50, 2017. [Online]. Available: https://doi.org/10.1063/PT.3.3626

[9] K. Barnes, P. Battaglino, B. J. Bloom, K. Cassella, R. Coxe, N. Crisosto, J. P. King, S. S. Kondov, K. Kotru, S. C. Larsen *et al.*, "Assembly and coherent control of a register of nuclear spin qubits," *Nature Communications*, vol. 13, no. 1, pp. 1–10, 2022.

[10] L. M. K. Vandersypen and M. A. Eriksson, "Quantum computing with semiconductor spins," *Physics Today*, vol. 72, no. 8, pp. 38–45, 2019. [Online]. Available: https://doi.org/10.1063/PT.3.4270

[11] A. Zwerver, T. Krähenmann, T. Watson, L. Lampert, H. C. George, R. Pillarisetty, S. Bojarski, P. Amin, S. Amitonov, J. Boter *et al.*, "Qubits made by advanced semiconductor manufacturing," *Nature Electronics*, vol. 5, no. 3, pp. 184–190, 2022.

[12] M. H. Devoret, A. Wallraff, and J. M. Martinis, "Superconducting qubits: A short review," *arXiv preprint cond-mat/0411174*, 2004.

[13] J. Koch, M. Y. Terri, J. Gambetta, A. A. Houck, D. I. Schuster, J. Majer, A. Blais, M. H. Devoret, S. M. Girvin, and R. J. Schoelkopf, "Charge-insensitive qubit design derived from the cooper pair box," *Physical Review A*, vol. 76, no. 4, p. 042319, 2007.

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

[15] S. H. Adachi and M. P. Henderson, "Application of quantum annealing to training of deep neural networks," *arXiv preprint arXiv:1510.06356*, 2015.

[16] D-Wave Systems Inc., "What is quantum annealing?" Jun 2022. [Online]. Available: https://docs.dwavesys.com/docs/latest/c_gs_2.html

[17] C. McGeoch and P. Farré, "The d-wave advantage system: An overview," *D-Wave Systems Inc., Burnaby, BC, Canada, Tech. Rep*, 2020.

[18] R. Hackett, "Ibm plans a huge leap in superfast quantum computing by 2023," Sep 2020. [Online]. Available: https://fortune.com/2020/09/15/ibm-quantum-computer-1-million-qubits-by-2030/

[19] Qiskit Development Team, "Introduction to transmon physics," Jun 2022. [Online]. Available: https://qiskit.org/textbook/ch-quantum-hardware/transmon-physics.html

[20] A. W. Cross, L. S. Bishop, S. Sheldon, P. D. Nation, and J. M. Gambetta, "Validating quantum computers using randomized model circuits," *Physical Review A*, vol. 100, no. 3, p. 032328, 2019.

[21] S. Aaronson and L. Chen, "Complexity-theoretic foundations of quantum supremacy experiments," *arXiv preprint arXiv:1612.05903*, 2016.

[22] A. Wack, H. Paik, A. Javadi-Abhari, P. Jurcevic, I. Faro, J. M. Gambetta, and B. R. Johnson, "Quality, speed, and scale: three key attributes to measure the performance of near-term quantum computers," *arXiv preprint arXiv:2110.14108*, 2021.

[23] Qiskit Development Team, "Ibm quantum lab tutorials overview," 2021. [Online]. Available: https://quantum-computing.ibm.com/lab/docs/iql/tutorials-overview

[24] ——, "Qiskit 0.37.0," Jun 2022. [Online]. Available: https://github.com/Qiskit

[25] V. V. Shende, S. S. Bullock, and I. L. Markov, "Synthesis of quantum-logic circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, pp. 1000–1010, 2006.

[26] D. McKay, G. Aleksandrowicz, and C. J. Wood, "Quantum tomography overview," Mar 2019. [Online]. Available: https://github.com/qiskit-community/qiskit-community-tutorials/blob/master/ignis/tomography-overview.ipynb?short_path=14a6c66

[27] C. Granade, C. Ferrie, and S. T. Flammia, "Practical adaptive quantum tomography," *New Journal of Physics*, vol. 19, no. 11, p. 113017, 2017.

[28] J. Gambetta, I. Faro, and K. Wehden, "Ibm's roadmap to build an open quantum software ecosystem," May 2022. [Online]. Available: https://research.ibm.com/blog/quantum-development-roadmap

[29] M. Y. Siraichi, V. F. d. Santos, C. Collange, and F. M. Q. Pereira, "Qubit allocation," in *Proceedings of the 2018 International Symposium on Code Generation and Optimization*, 2018, pp. 113–125.

[30] G. Li, Y. Ding, and Y. Xie, "Tackling the qubit mapping problem for nisq-era quantum devices," in *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*, 2019, pp. 1001–1014.

[31] S. Jandura, "Improving a quantum compiler," Sep 2018. [Online]. Available: https://medium.com/qiskit/improving-a-quantum-compiler-48410d7a7084

[32] C. Zhang, A. B. Hayes, L. Qiu, Y. Jin, Y. Chen, and E. Z. Zhang, "Time-optimal qubit mapping," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 360–374.

[33] P. Murali, J. M. Baker, A. Javadi-Abhari, F. T. Chong, and M. Martonosi, "Noise-adaptive compiler mappings for noisy intermediate-scale quantum computers," in *Proceedings of the twenty-fourth international conference on architectural support for programming languages and operating systems*, 2019, pp. 1015–1029.

[34] F. B. Maciejewski, Z. Zimborás, and M. Oszmaniec, "Mitigation of readout noise in near-term quantum devices by classical post-processing based on detector tomography," *Quantum*, vol. 4, p. 257, 2020.

[35] E. Riedel Gårding, N. Schwaller, C. L. Chan, S. Y. Chang, S. Bosch, F. Gessler, W. R. Laborde, J. N. Hernandez, X. Si, M.-A. Dupertuis *et al.*, "Bell diagonal and werner state generation: entanglement, non-locality, steering and discord on the ibm quantum computer," *Entropy*, vol. 23, no. 7, p. 797, 2021.

[36] M. B. Pozzobom and J. Maziero, "Preparing tunable bell-diagonal states on a quantum computer," *Quantum Information Processing*, vol. 18, no. 5, pp. 1–12, 2019.

[37] A. Cervera-Lierta, "Exact ising model simulation on a quantum computer," *Quantum*, vol. 2, p. 114, 2018.

[38] F. Verstraete, J. I. Cirac, and J. I. Latorre, "Quantum circuits for strongly correlated quantum systems," *Physical Review A*, vol. 79, no. 3, p. 032316, 2009.

[39] E. Lieb, T. Schultz, and D. Mattis, "Two soluble models of an antiferromagnetic chain," *Annals of Physics*, vol. 16, no. 3, pp. 407–466, 1961.

[40] S. Katsura, "Statistical mechanics of the anisotropic linear heisenberg model," *Physical Review*, vol. 127, no. 5, p. 1508, 1962.

[41] P. Jordan and E. Wigner, "Über das paulische äquivalenzverbot," *Zeitschrift für Physik*, vol. 47, pp. 631–651, 1928.

[42] A. Smith, M. Kim, F. Pollmann, and J. Knolle, "Simulating quantum many-body dynamics on a current digital quantum computer," *npj Quantum Information*, vol. 5, no. 1, pp. 1–13, 2019.

[43] N. Hatano and M. Suzuki, "Finding exponential product formulas of higher orders," in *Quantum annealing and other optimization methods.* Springer, 2005, pp. 37–68.

Erklärung

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und dabei keine anderen als die angegebenen Quellen verwendet habe. Sämtliche Bilder, Ideen und Ergebnisse aus fremder Quelle habe ich als solche kenntlich gemacht. Die Arbeit wurde bisher weder gesamt noch in Teilen einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Siegen, 03. August 2022, ...

Unterschrift