Institute for Theoretical Physics
University of Cologne

Prof. Dr. S. Trebst
Q. Preiss, F. Eckstein, M. Pütz

Quantum Computational Physics

Exercise Sheet 7

Winter Term 2024/25

Due date: Friday, 24.01.2025 **Discussion**: Friday, 17.01.2025

Website: thp.uni-koeln.de/trebst/Lectures/2024-QuantCompPhys.shtml

Exercise 11: QuantumClifford.jl

Quantum circuits that only consist of Clifford gates and measurements can be efficiently simulated using the **stabilizer tableau formalism**. Such a simulation stores the state of an n qubit circuit using a $n \times 2n+1$ matrix of binary values (tableau) that encodes stabilizer generators. The stabilizer generators encode the state in the sense that the encoded state is a common eigenvector of all stabilizer generators with eigenvalue 1. Each Clifford gate can then be represented by a simple update rule on the tableau.

There are some nice packages out there that provide a simple interface to implement such stabilizer tableau simulations. `QuantumClifford.jl` is the Julia package of choice for us. First install the package by running the following command in the Julia REPL:

julia
```
using Pkg
Pkg.add("QuantumClifford")
```

Then you can use the package by running:

julia
```
using QuantumClifford
```

To create a simple tableau you can use the convenient string constructor. For example, the following code creates a tableau that represents the state $|000\rangle$:

julia
```
tableau = S"+Z__
    +_Z_
    +__Z"
```

The `_` represents the identity operator and the `Z` the Pauli-Z operator. Other operators can be used as well, such as `X` and `Y`. The `+` represents the phase and can be one of `+`, `-`, `+i`, `-i`. The given tableau represents the state $|000\rangle$, because $|000\rangle$ is the only state that is a common eigenvector of the stabilizer generators ZII, IZI and IIZ with eigenvalue 1. In `QuantumClifford.jl` the type of the tableau created above is called `Stabilizer`.

Clifford gates can be constructed similarly using the string constructor. For example, the following code creates a Pauli-X gate:

julia
```
pauli_X = C"+X
    -Z"
```

Here `pauli_X` is of type `CliffordOperator`. It indicates a mapping of how the stabilizer changes when the Pauli-X gate is applied to the state. Namely, $X_1 \to +X, Z_1 \to -Z$. Or $XXX^\dagger = X$ and $XZX^\dagger = -Z$. To apply a Clifford gate to a stabilizer, you can use the `apply!` function. For example, the following code applies the Pauli-X gate to the tableau created above:

julia
```
apply!(tableau, pauli_X, [1])
```

The third argument of the `apply!` function is a list of qubit indices that the gate is applied to. Since the Pauli-X gate is a single qubit gate, the list only contains one element. As a result, the generators are now `-Z__`, `+_Z_` and `+__Z` which represent the state $|100\rangle$ as expected.

a) Create a tableau that represents the state $|+++\rangle$.

b) Create the remaining Pauli operators Y and Z as well as the Hadamard gate using the string constructor.

Measurements are a bit more complicated in the tableau formalism, but are still simple to implement using `QuantumClifford.jl`. A measurement always measures a Pauli string, i.e., a Pauli operator on each qubit. The Pauli string can again be easily constructed using the string constructor. For example, the following code constructs a Pauli string that measures the Z operator on the first qubit:

julia
```
pauli_string = P"+Z___"
```

Now to measure this Pauli string on a state, you can use the `projectrand!` function. For example, the following code measures the first qubit of a four qubit GHZ state in the Z basis:

julia
```
ghz_tableau = S"+XXXX
    +ZZ__
    +_ZZ_
    +__ZZ"
projectrand!(ghz_tableau, pauli_string)
```

This function returns the new tableau (`Stabilizer`) after the measurement, as well as the randomly chosen phase of the measurement outcome (that's why the name of the function contains `rand`).

c) Implement a parity measurement in the Z basis on the first two qubits of the GHZ state.

d) Implement the glassy GHZ state preparation from exercise 4 using the `QuantumClifford.jl` package. Can you tell from the tableau that the state is a glassy GHZ state?

e) Complete the GHZ state preparation by storing the measurement results and applying the necessary corrections to the state as in exercise 4. Do you recognize the state now from the tableau?

f) ($\star$) To check if the state is indeed a GHZ state, you could construct the density matrix and since the state should be pure, find the eigenvector with eigenvalue 1.
Hint: The density matrix ρ can be constructed using

$$\rho = \prod_{i=1}^{n} \frac{I + g_i}{2} \tag{1}$$

where g_i are the stabilizer generators of the state.

To speed up the simulation of measurements, `QuantumClifford.jl` provides a different type called, `Destabilizer`. At the cost of keeping track of more information (destabilizers), the measurement time complexity is reduced from $O(n^3)$ to $O(n^2)$. Even better is the `MixedDestabilizer` type, which also speeds up simulations that simulate mixed states, which are encoded by storing less than n stabilizers in the tableau. You can construct a `MixedDestabilizer` from a `Stabilizer` as follows:

julia
```
mixed_destabilizer_tableau = MixedDestabilizer(tableau)
```

`QuantumClifford.jl` also provides a function to compute the entanglement entropy between a subsystem and the rest of the system.

julia
```
entanglement_entropy(state, qubits_in_subsystem, Val(:rref); pure=true)
```

This will be useful in the next exercises, when we want to investigate different entanglement structures.

g) For now just compute the entanglement entropy of the GHZ state from the previous exercise. Is the GHZ state maximally entangled?

Exercise 12: Entanglement pyramids

Now that we understand the basics of stabilizer tableau simulations and how to program them using the `QuantumClifford.jl` package in Julia, we want to explore the formation of entanglement pyramids and evidence of KPZ physics in random unitary circuits as it was discussed in the lecture.

The specific circuit we want to simulate for this purpose is composed of multiple layers of two qubit gates, which are applied on each bond of a periodic 1D chain of qubits (in a "brickwall pattern" with each layer first odd, then even bonds – as depicted in Fig. 1 below). These two qubits gates are randomly chosen from the set of all Clifford gates. In `QuantumClifford.jl` you can use the function

julia
```
random_clifford(2)
```

where 2 is the number of qubits the gates act on.

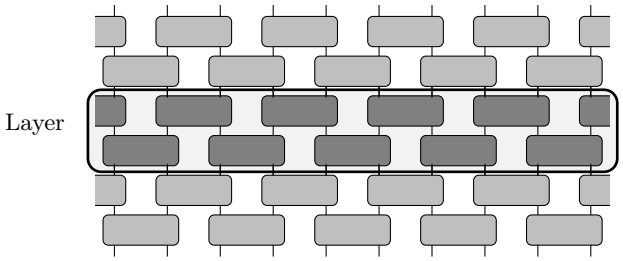

Figure 1 – A snippet of the deep random circuit. The unitary two-qubit gates (gray boxes) are randomly chosen from the Clifford group.

a) Initialize a Stabilizer in the Z basis which represents the initial state of the qubit chain, given a length/amount of bonds in the chain.

b) Write a function to simulate t layers of the random circuit starting from the initial state constructed in the previous part.

c) Compute the entanglement entropy across every bond of the chain after t layers of the circuit and plot the result as a function of the bond index. Repeat this for different values of t and compare the results. Average your results to reduce the noise created by the randomness of the system.
Hint: This should result in the expected shape of the KPZ scaling pyramid. A chain of 460 qubits simulated for some 200 layers should give good results. When averaging over about 100 circuit realizations, expect a runtime around 5 mins.

Exercise 13: Measurement-induced entanglement transition

In this exercise we want to add (non-unitary) measurements to the mix of random operations in our circuit and investigate whether this affects the entanglement structure, such as the entanglement pyramid produced by random unitary gates only. Is there an "entanglement phase transition"?

Specifically, the idea is to intersperse the circuit we looked at in the previous exercise with singe-qubit projective measurements in, say, the Z-basis. Let us vary the amount of such interspersed measurements by introducing a measurement probability p, which we want to slowly increase to transition from $p = 0$, i.e. a random unitary circuit giving rise to volume-law entanglement as we found out above, to a situation with lots of measurement where we no longer expect the volume-law entanglement to survive. But what is it replaced by? And for what probability/strength of measurements does this happen?

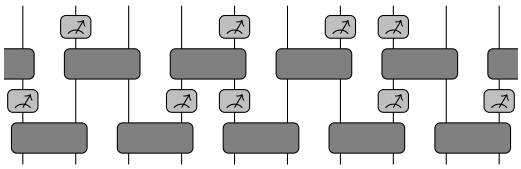

Figure 2 – A layer of the deep circuit with random unitaries and interspersed measurements. The two-qubit unitary gates (gray boxs) are again randomly chosen from the Clifford group, while (after every row of two qubit gates) we add random single-qubit measurements (in the Z basis) with a probability p.

a) Add single-qubit measurements to your circuit from the previous exercise. After every layer of two qubit unitaries, each qubit should be measured with a measurement probability p (see fig. 3), i.e. every qubit has the chance to be measured twice per layer.

b) Run the same simulations as in exercise 12 c) but for the measurement circuit created in part a) of this exercise for different probabilities $0 \leq p \leq 1$.

c) Plot the entropies at the last time step for different probabilities into one plot on a log-log scale. Do you see the different entanglement regimes (phases) and a phase transition in between? How can you locate the transition? *Hint:* Can you see the three different entanglement scaling regimes sketched in the figure below?

d) Can you motivate what happens in the two distinct regimes and at the transition?

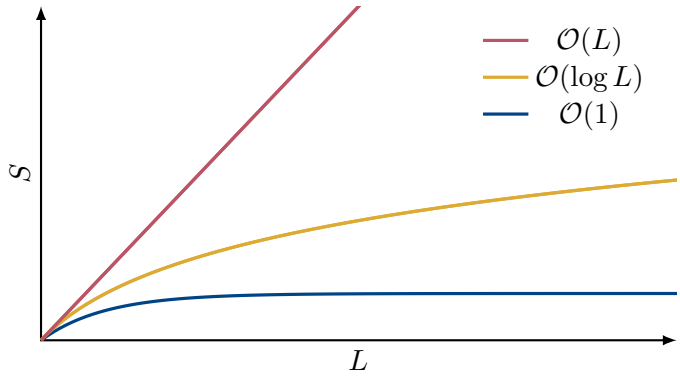

Figure 3 – Scaling of the entanglement entropy for different entanglement phases. For a small amount of measurements ($p < p_c$), the entanglement creation outperforms the entanglement reduction due to the measurements, resulting in a volume law scaling (red). For a large amount of measurements ($p > p_c$), the entanglement reduction dominates, resulting in an area law scaling (blue). At the critical point, the entanglement creation and reduction are balanced, resulting in a logarithmic scaling (yellow).