README

Sun's Algorithm Implementation

Andrea Bersellini

February 2025

1 Description

This project consists of the implementation of Sun's Algorithm for the problem of quantum state preparation of n qubits, using $2n$ ancillary qubits. This represents a particular subcase of the range presented in the first theorem of the paper.

All modules have been developed in Python with the support of the PennyLane library, which enables quantum computing simulations.

The project includes various files containing classes and functions essential for the complete implementation:

- **functions.py:** Contains all the functions used to operate with vectors, print results, and compute parameters using traditional computing algorithms.
- **quantum_circuit_classes.py:** Contains all the classes used to generate the desired quantum circuit and to simulate quantum states.
- **lambda.py:** Contains the implementation of the Λ_n circuit, described in the original paper, with n as a variable, using an initial random complex vector.
- **qsp_n2_random.py:** Contains the implementation of the whole quantum state preparation circuit with $n = 2$, using a random complex vector as the desired final quantum state.
- **qsp_bell_states.py:** Contains the implementation of the whole quantum state preparation circuit with $n = 2$, using a specific Bell state as the desired final quantum state.

2 Requirements

The project has been implemented using Python 3.12 (Python 3.10 or later should work as well) and the latest version of the *PennyLane* library. The implementation also makes use of the *math*, *operator*, *numpy*, and *matplotlib.pyplot* modules, which should already be present in the latest version of Python.

3 Usage

To test the implementation, it is sufficient to run *"qsp_n2_random.py"* to generate a random desired 2-qubit quantum state. It is also possible to generate a specific, real or complex, 2-qubit quantum state by modifying the *coefficients* variable with a 2^n-element vector of l_2-norm = 1:

```
coefficients = complex_numbers.tolist() # Modify this to insert the
    desired vector
```

3.1 Quantum Circuit Implementation

Every implemented circuit makes use of some derived classes from the abstract class *Stage*, to define the position of the gates and the values of some useful parameters, and instantiate a *QuantumCircuit* object that is basically a collection of stages and allows displaying the circuit and the quantum state in various ways using the `QuantumCircuit.printCircuit(params)` method.

3.2 Calculation of Parameters

Multiple functions are used to calculate the parameters needed to induce all the phase shifts in the computational basis; this process is done using traditional computation and can therefore be executed separately from the quantum computation part. A user can simply call the `qspParameters(coeff_vector, num_of_qubits)` function to generate $(2^n - 1)$ α angles from the desired coefficient vector.

3.3 Λ Circuit

With the current implementation, it is possible to build the circuit for Λ_n, with n as a variable, thanks to the adopted stage structure.

The actual script works by initializing the variable n and then randomly generating the complex coefficient vector, whose modulus corresponds to the desired final quantum state. Then, the α values are calculated and associated with each computational basis string. Finally, using the *Stage* classes, the quantum circuit is constructed, and the final state —in this case, the values of the diagonal of the matrix associated with each multiplexor— can be retrieved.

4 Unipr HPC Server

When using the HPC server to run the script, it is recommended to create a new virtual environment with the required version of Python. Additionally, it may be necessary to disable the circuit print command:

```
circuit.printCircuit(mode="figure", modulo=False) # Comment this
```

as it opens an interactive window, which could stall the execution of the script indefinitely, and to change the input from console for the same reason. It is also possible to specify some parameters in the script "*.sh*" to schedule the execution and save the outputs:

```
--job-name = quantum_state_preparation
--output = %x.o%j
--error = %x.e%j
--nodes = 1
--partition = cpu
--qos = cpu
--time = 0-00:10:0
```