Partial report verification/testing

# Verification

Verification of the core engine is an important factor in ensuring the quality of the implementation. Although some parts of the system have undergone automated self-testing based on unit tests, the overall performance needs to be benchmarked against well-established results. Quantitative results should objectively show the fitness of the system.

This being said, verification of the current system as a whole is a quite challenging task. The reason is that this is the first full MBQC simulator that allows simulation of MCalc patterns. The lack of such software means that only parts of the whole could be automatically tested. Nevertheless several solutions exist that could aid the testing process.

The methods that are presented in this section use already existing software that has been independently verified. They are based on the assumption that the third party software is producing correct results. Those methods could be in theory automated to produce consistent outputs and reliable benchmark for accuracy. This would guarantee that the components that are undergoing the aforementioned procedures are free from bugs in the context of the data that is being analysed. No testing could possibly guarantee absolutely bug free performance unless all possible inputs/output pairs of the algorithm are known in advance. This is impractical in the context of a simulator.

## Experimental Data Analysis

The first approach to verifying the fitness of the implementation is to actually use real experimental data and compare it with the predictions done by the simulator. The experimental data that is currently available is taken from an experiment on optical implementation of quantum computing. The choice of data source is rather useful since there already exists a third party software that is specifically developed to analyse this particular experimental setup. Furthermore, the available software has been tested and provides a good benchmark for performance and accuracy.

### Optical Implementation of Quantum Computing

The experimental setup is described in details in *Diagnosing Optical Implementations of Quantum Computing* by Flaviu Cipcigan. In a nutshell, a laser produces photons which pass through a series of optical filters. Each filter applies a certain quantum operation to the photons. The original stream is split into several photon beams. Detectors at the end of the optical paths measure the arrival of individual photons. Most of the events are discarded, since the probability of entangling photons is quite low. When photons arrive at the same time in all of the detectors, a certain assumption could be made about the particles and their quantum states. Events that have such coincidences are recorded and the number of photons arriving at those states is stored into a csv file.

The quantum “algorithm” implemented could be neatly described in terms of MCalc operations onto an initial system called a “lab cluster”. A simulation of this lab cluster with a series of measurements could be done. This will change the state of the cluster. The probability of obtaining this particular state could be calculated and it directly relates with the relative number of photons that are expected to trigger the simultaneous events at the detector. The predicted probability and the actually obtained one could be compared and the quality of the comparison will prove the quality of the simulation algorithm.

The lab cluster contains 4 qubits. They start in an initial state

Where *n* depends on the type of experiment that is being performed and is recorder in the filename of the csv file.

A series of 4 measurements are made, which could be either an X, Y, M, P or Z measurement. Each set of measurement corresponds to a row in the csv file. For each of them, events are recorded for all of the possible 4 branches thus generating 16 columns for each row.

In order to obtain the probability of a particle ending up in a certain branch when a certain sequence of measurements is applied, the state of the lab cluster is used. As mentioned previously in the report, the probability of a branch is the norm squared of the resulting vector.

### Simulation

In order to implement a simulator tailored for the optical quantum computation experiment, several additional entities needed to be added to the already existing system.

**LabCluster** – a class that extends SystemMatrix and allows for initialization of the pool of bits to a certain lab cluster. Once an instance of this class is created, measurements could be directly applied to it and the probability could be extracted using the inherited methods from SystemMatrix.

**ZM** – a Z measurement in the basis of . This literally means that a Z measurement in branch 0 will take only the coefficients of the current qubit that are in the state, setting the others to 0. A measurement in branch 1 will take only the coefficients in the state.

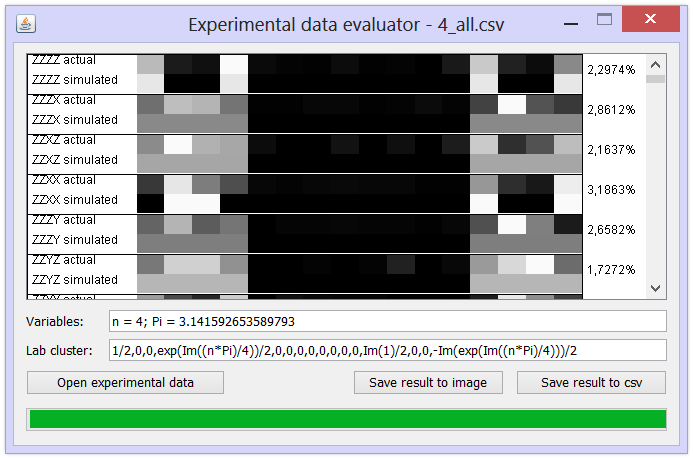
**WorkSheet, TableStorage** – for parsing the csv and storing the original experimental probabilities for each pair of measurements and branches.

**ExperimentVisualizer, CSVExporter** – for converting predicted output to images and csv files for further analysis and comparison with the original data.

**ExperimTest** – performs the tests by loading a file. It takes an ASCII version of the initial cluster state and runs an experiment. Some sanity checks of the results and inputs is also done and warnings are issued in cases in which total probability does not add up to 1 or input is not normalized. A callback ensures that updates are being passed to the original invoker, allowing for user friendly features like a progress bar to be rendered on the screen. It then returns a result that contains a CSV file and an image as a result of the simulation.

A GUI interface was developed for intuitively interacting with the simulation. It allows for defining a custom lab cluster and setting values for variables. It then shows the visual representation of the analysis on screen and gives option to save it as an image file or as a csv file. Also a progress bar is shown while the analysis is taking place so the user is aware of the status of the computation.

This configuration allows for flexibility in the simulation. If a new set of experiments is generated (that follows the same input csv format), the analysis could be easily adapted by changing the initial state of the lab cluster. The system can handle an arbitrary number of qubits as well (i.e. the lab cluster could in theory hold more than 4 qubits) as long as the number of qubits in the csv file matches with the number of qubits in the lab cluster.



Simulation output

Cluster definition

Input/output

### Output

Figure 1 – GUI of the tool for experimental data evaluation and comparison

The output from the simulator could be best visualized using a grayscale mapping[[1]](#footnote-1). The mapping is between the minimum (black) and maximum (white) probability in each row between the actual and the simulated data.

min

max

How to interpret a “row” from the generated result image:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Measurement sequence | Branches | | | | | | | | | | | | | | | | Average residual error |
| 0 0 0 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 1 0 0 | 0 1 0 1 | 0 1 1 0 | 0 1 1 1 | 1 0 0 0 | 1 0 0 1 | 1 0 1 0 | 1 0 1 1 | 1 1 0 0 | 1 1 0 1 | 1 1 1 0 | 1 1 1 1 |



The row above means that for each cell, which is illuminated, the lab cluster needs to be initialized and 4 measurements must be done (in this case a Y measurement, followed by three consecutive Z measurements). The first column represents all measurements resulting in a projection to the 0 branch. The second column means the final Z measurement is resulting in a projection to the 1 branch while the other 3 result to the 0 branch.

Therefore for each row like this, 16 simulations are done using the following algorithm:

*For* each branch combination **B** in **bs** *do*:

*Initialize* new lab cluster

*For* **i** from 0 to size of **ms** do:

*Perform* measurement **ms**[**i**] taking a branch **B**[**i**]

*Calculate* probability of the cluster

For the example above, **ms** = {Y, Z, Z, Z}, **bs** = { {0,0,0,0}, {0,0,0,1}, {0,0,1,0} … {1,1,1,1} }

If we assume that we have the original probabilities in an array **orig** and the simulated probabilities in an array **sim**, then we could take the values **max\_val** = *max*( *max*( **orig** ), *max* ( **sim** ) ) and **min\_val** = *min*( *min*( **orig** ), *min* ( **sim** ) ). The colour for each cell is calculated as **colour** = (**current\_value** – **min\_val**) / ( **max\_val** – **min\_val**). The resulting value is mapped as **colour** = 0.0 means *black* and **colour** = 1.0 means *white*.

The average residual error is the average absolute difference between results obtained in the experiment and the simulated output for a given row. It gives a quantitative measure on the accuracy of the theoretical fit. It is reported in percent, which is the actual value multiplied by 100.

The relationship of actual values vs predicted vs residuals should be clear from clear from the figure above. The average value of the residual line is 0.0166 therefore the reported value as an average residual error for the row shown in Figure 1 is 1.6617%. You could also see that in this case **max\_val** = 0.149 and **min\_val** = 0.

### Results of verification

The verification has been done on 8 lab clusters, each with 226 measurement tuples, each with 16 possible branches. This totals to 28928 simulations for the available data.

Average residual of all: 0.03684536664699281

Max 0.10810810886323452 in ZMXY in file 6\_all.csv

Min 0.008804020937532187 in XYMX in file 1\_all.csv

1. Flaviu Cipcigan, Diagnosing Optical Implementations of Quantum Computing [↑](#footnote-ref-1)