Partial report verification/testing

1 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.

1.1 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.

1.1.1 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

$$\frac{1}{2}[|0000\rangle + e^{i\frac{n\pi}{4}}|0011\rangle + i|1100\rangle - ie^{i\frac{n\pi}{4}}|1111\rangle]$$

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.

1.1.2 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.

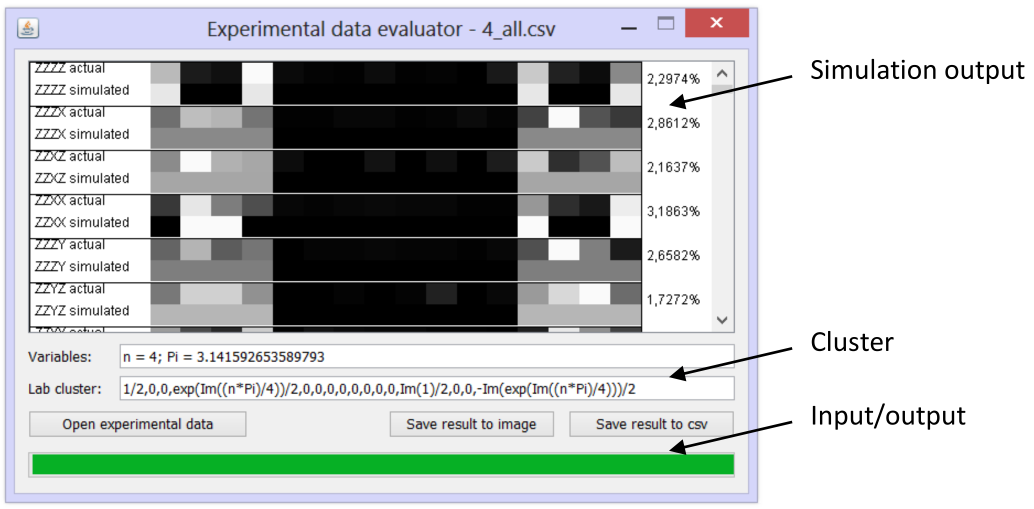
ZM – a Z measurement in the basis of $\{|0\rangle, |1\rangle\}$. This literally means that a Z measurement in branch 0 will take only the coefficients of the current qubit that are in the $|0\rangle$ state, setting the others to 0. A measurement in branch 1 will take only the coefficients in the $|1\rangle$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



ZZZZ actual
ZZZZ simulated 2,2974%
ZZZX actual
ZZZX simulated 2,8612%
ZZXZ actual
ZZXZ simulated 2,1637%
ZZXX actual
ZZXX simulated 3,1863%
ZZZY actual
ZZZY simulated 2,6582%
ZZYZ actual
ZZYZ simulated 1,7272%

Variables: n = 4; Pi = 3.141592653589793

Lab cluster: 1/2,0,0,exp(Im((n*Pi)/4))/2,0,0,0,0,0,0,0,0,Im(1)/2,0,0,-Im(exp(Im((n*Pi)/4)))/2

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.

1.1.3 Output

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

How to interpret a "row" from the generated result image:

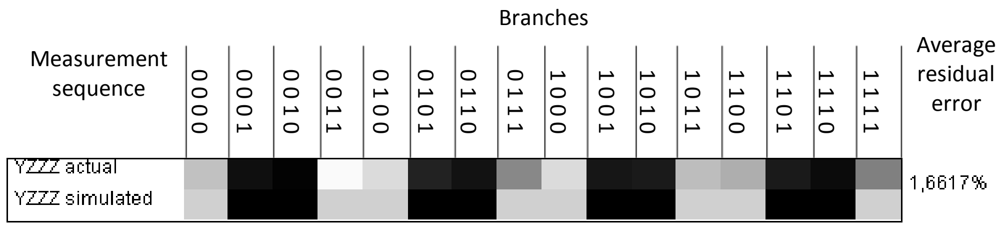


Figure 1 – GUI of the tool for experimental data evaluation and comparison

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:

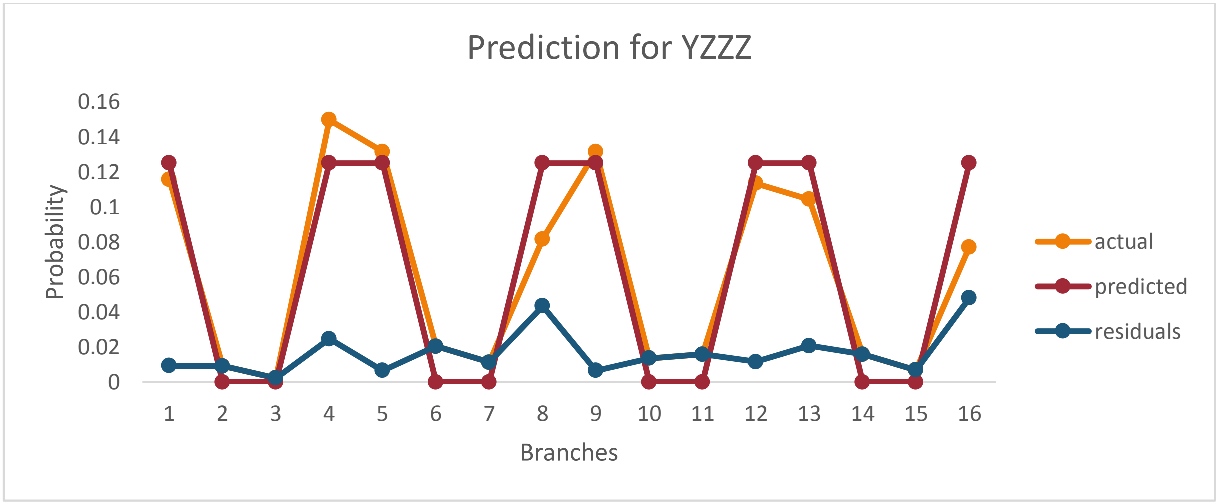
[1] Flaviu Cipcigan, Diagnosing Optical Implementations of Quantum Computing

```
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.

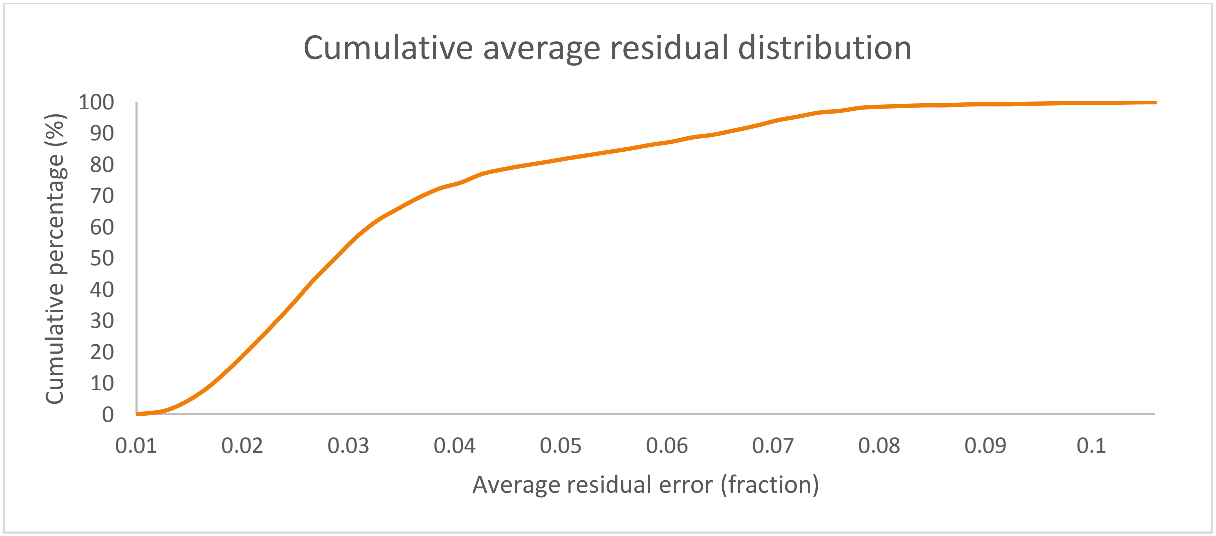


The relationship of actual values vs predicted vs residuals should be clear from clear from the figure above. The average value of the residual line is 0.0166 therefore the reported value as an average residual error for the row shown in Figure 1 – GUI of the tool for experimental data evaluation and comparisonis 1.6617%. You could also see that in this case **max_val** = 0.149 and **min_val** = 0.

1.1.4 Results of verification

The verification has been done on 8 lab clusters, each with 225 measurement tuples, each with 16 possible branches. This totals to 28800 simulations for the available data. The average residual error is 3.68%. The best simulated experiment has a residual error of 0.8% while the worst one has an error of 10.8%.

49.83% of the rows have a value of the average residual error less than 2.96%. Only 27.61% of the data has an average residual error bigger than 3.95%. This could be clearly seen on the cumulative average residual distribution profile below. The plot shows the fraction of data rows which have an average residual error less than a certain value.



This clearly show that majority of the data could be explained using the simulation correctly. Here are two examples of a good and a bad theoretical prediction.

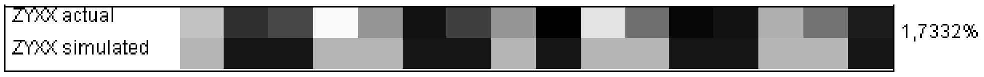


Figure 2 Good agreement with simulation

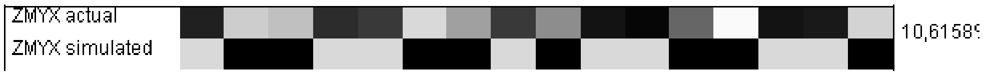
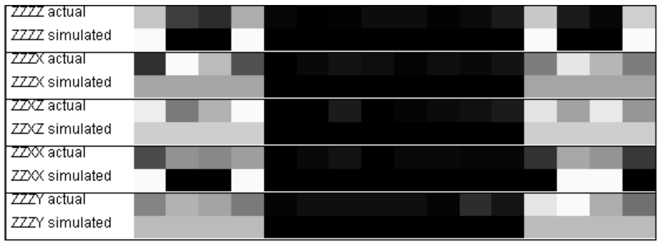


Figure 3 Disagreement with simulation

It could be seen that the actual data is quite noisy but the pattern that it follows could be observed. The reason for disagreement is the Y measurement and it is explained in detail in *Diagnosing Optical Implementations of Quantum Computing* by Flaviu Cipcigan.

A result of the *Diagnosing Optical Implementations of Quantum Computing*, a software was developed for specifically verifying the results of the above experiment. The software was run on the exact data set input files and the predicted



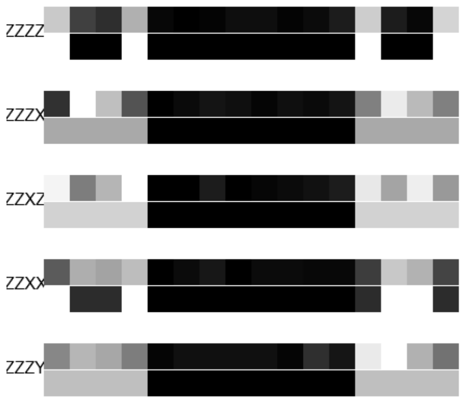


Figure 4 Comparison between output of the current project (on the left) and Mr Cipcigan's project (on the right)

The resulting data was compared and the two applications agreed completely on the predicted values. This could be seen on Figure 4 Comparison between output of the current project (on the left) and Mr Cipcigan's project (on the right). These are the first 5 measurements of the 7th lab cluster. They match completely as do all of the other predictions.

1.1.5 Conclusion

It should be made clear that Mr Cipcigan's software uses a completely different algorithm for generating the theoretical predictions. His software works exclusively with probabilities. The core of the current project is a full MBQC simulation. Furthermore, the current project uses symbolic algebra which could make printing the exact probability and lab cluster more human readable.

It is apparent from the results presented in the previous subsection that the output from the current simulator completely matches with the output from an already existing software. Furthermore, it could be successfully used to explain and analyze experimental data. Therefore this should guarantee the correctness of the implementation of several of the core systems – the symbolic mathematics manipulation and the MCalc and cluster operation. This is the essentially all of the required components for a full MBQC simulator.

It should be noted that the test above does not test the user-interaction part of the system. Systems that are not being used by the above test are the ASCII to MCalc translation and the graphical pattern to MCalc translation. Those systems would be targeted with the next verification technique.

1.2 RUNNING ARBITRARY MBQC PATTERNS

Simulation of an arbitrary MBQC pattern was not possible before the development of this software. Therefore directly comparing the output of a random MBQC pattern with an already existing software solution is impossible. This makes the task of evaluating the accuracy of the ASCII and graphical to MCalc translation challenging. Several approaches could be taken in order to verify the correctness of the implementation.

1.2.1 Semi-automated Procedure

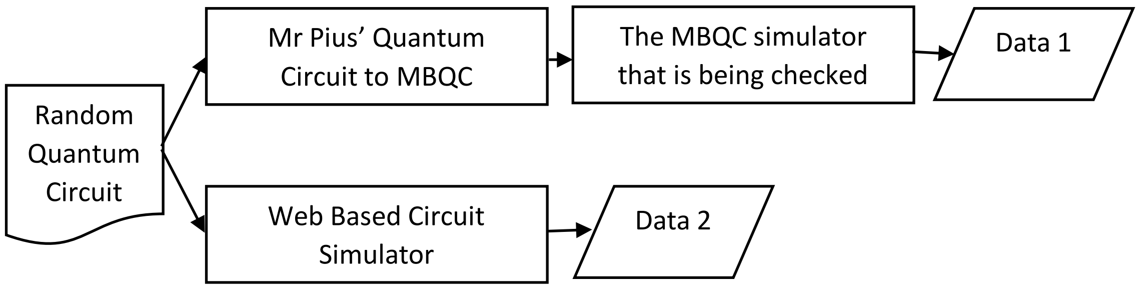
The idea is to use more than one already existing software packages and pipe their inputs and outputs together. The method proposed below, in theory, could be fully automated to generate a good test. The reason the automation hasn't been done at this point is that one of the third party software does not have an automated input system but rather relies on a GUI.

The software packages that have been used are:

- Software[2] developed by Einar Pius that translates quantum circuits into MCalc description. The software is written in C++.
- Quantum Circuit Simulator[3] by Davy Wybiral that simulates quantum circuits. The software is web based and written in JavaScript.

[2] Einar Pius, Automatic Parallelisation of Quantum Circuits Using the Measurement Based Quantum Computing Model
[3] http://www.davyw.com/quantum

The way the procedure works:


Random Quantum Circuit
Mr Pius' Quantum Circuit to MBQC
The MBQC simulator that is being checked
Data 1
Web Based Circuit Simulator
Data 2

The Random Quantum Circuit could be literally a random generated one. It does not need to represent anything physically meaningful. The verification method will produce two sets of data **Data 1** and **Data 2** for each Quantum Circuit. Those two sets could be compared in order to evaluate the correctness of the implementation.

In order to utilize the method above, the links between the methods should be implemented. In theory, all of these could be automated. Currently they need to be done by hand, since the web based simulator cannot be interfaced easily with the other components. Furthermore the format of the quantum circuits that Mr Pius' translator uses is not completely compatible with the Web Based Simulator. Some translation needs to be done manually between those two formats.

The main difference is the quantum gates that the two software packages use are different. Mr Pius has kindly provided his assistance for clarifying the process of translating the input of his application to be the same as the input that the Web Based Circuit Simulator would expect.

Furthermore the MCalc descriptions generated by Mr Pius' program are not always compatible with the MCalc simulator. This is because there are several restrictions in place. One physical one is that more than 13 – 14 qubits are impractical to simulate because it would require storing 2^{13-14} coefficients. Storing them and working with them in general would be extremely slow. Therefore MCalc patterns that have this number of bits are infeasible to simulate. Furthermore, the current GUI assumes input qubits could be only the first *n* qubits from a pattern. This is not a limitation of the underlying engine, but rather a limitation of the translation algorithm. Therefore patterns that don't obey this need to be manually translated.

Despite the difficulties, to prove that the outlined procedure is a viable way of testing, several completely random Quantum Circuits were generated. They were fed into the Mr Pius' software and a compatible version was fed into the web based simulator. The MCalc result from Mr Pius' software was then used as input for the MBQC simulator.

Quantum Circuit	Circuit result	MCalc description	MCalc result for
ZZ q1 q2	-1.0 |11>	E 1 2	-1.0 |11>
ZZ q1 q2 ZZ q2 q3 H q3		E 1 2 E 2 3 E 3 4 M(0) 3; X 4; s = 3	