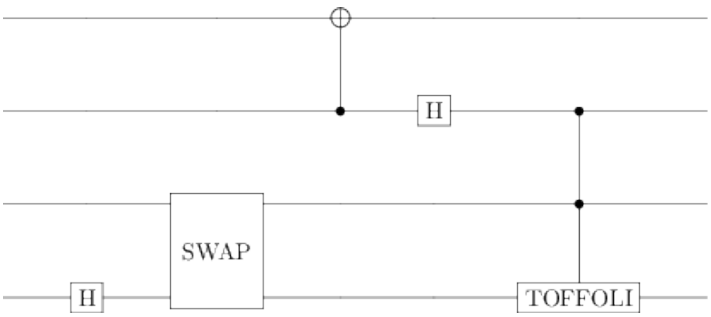

Tools for Quantum Circuit Synthesis

Chelsea Voss

6.845, Quantum Complexity Theory

Fall 2014

Motivation

Quantum circuits are one way to model the behavior of quantum computers. In a quantum circuit, a collection of input qubits is first passed through a series of gates (which may perform operations on those qubits), then measured to produce an output. Unlike classical circuits, the circuit must have as many wires at the output as it does at the input, since every operation must be unitary and therefore reversible and information-preserving. Also unlike classical circuits, each qubit may exist in a more complicated state than purely 0 or purely 1, and the output of the circuit may be probabilistic instead of deterministic.

There are many existing interfaces which model the behavior of classical circuits[1], some of which even treat the problem of designing a classical circuit as a puzzle game[2]. However, interfaces for quantum circuit design are relatively unexplored. jQuantum provides an applet to serve this purpose, although it does not seem to support many gates. In this project, I wanted to create a tool that would provide a flexible sandbox for quantum circuit design, allowing the use of many gates, allowing already-built circuits to be exported as subcircuits, and allowing the overall behavior of the circuit to be easily analyzed.

I designed and built a tool, named **QuantumMechanic**, that permits users to simulate small quantum circuits by adding quantum gates one by one. The primary purpose of this project is to aid researchers who need to construct quantum circuits as part of a particular proof or reduction; a secondary goal is to create puzzles for people who are interested in understanding quantum circuits better, so that they can learn by playing puzzles.

The code for this project can be found at <http://github.com/csvoss/quantummechanic>. The tool itself is currently live, and can be viewed at <http://18.247.0.45:8000/>.

UI Design

In order to allow users to simulate small quantum circuits, the following features are essential:

- Add gates to the circuit
- Display the circuit as an image
- Remove gates from the circuit
- Analyze the output of the circuit in at least one way

In addition, the following features would improve the usefulness or usability of the tool, while not being absolutely essential:

- Rearrange the order of gates
- Add, rearrange, or delete gates using a drag-and-drop interface
- Save a circuit as a sub-circuit to use in later designs
- View all possible outputs of the circuit, with weighted probabilities

1 Logicly, a logic circuit simulator: <http://logic.ly/>

2 *KOHCTPYKTOP, Engineer of the People* is one such game: <http://www.zachtronics.com/kohctpyktop-engineer-of-the-people/>

- Compare the circuit to another circuit (as the target of a puzzle, for example)

With this in mind, I designed a user interface that would, at the minimum offer the essential features, while leaving open the possibility of adding the extra features in the future.

Gates can be added to the circuit by clicking on them. Dropdown menus govern which qubit or qubits the gate is added to. Some gates require an additional parameter, theta, which can be specified by typing a number (in degrees) into the "Theta" box. Another box permits the user to specify where the gate is added. Finally, a delete button permits the deletion of gates from the circuit.

The current UI differs from the UI that I initially had in mind, because dragging and dropping gates proved to be more difficult than anticipated, and I prioritized getting all of the simple features in place before developing a more sophisticated user interface. For now, this simple interface provides enough to permit users to edit circuits.

Implementation

To construct this tool, first I needed a quantum circuit simulator. Of the many options[3] available, QuTIP[4] seemed to be the best option: written in Python, actively developed, and featuring simulation of various quantum gates.

I built a website using the Django web framework[5]. Django runs in Python, so interfacing with QuTIP becomes simple: I can import the library into any Python code I run. Django uses Python both to generate the webpage and to dynamically respond to requests with new content, so I am able to run QuTIP to generate the initial list of gates as well as whenever the user adds a new gate or checks the output of the circuit.

Whenever a user loads the QuantumMechanic sandbox page, the server is responsible for rendering images of each gate building-block as well as an image of the empty circuit in the initial HTML interface it generates. Next, whenever the user decides to add another gate, the client-side HTML page sends an Ajax request to the server, asking it to use QuTIP to add the gate and render a new image of the circuit-in-progress. A similar process takes place whenever a gate is deleted and whenever the user wants to analyze the output of the circuit. Each of these functionalities has a bit of Javascript code in the client-side HTML page dedicated to sending the relevant requests to the server, and a bit of Python code in the server dedicated to performing the computation and returning the result.

Challenges

QuTIP Installation and Use

3 Quantiki/List of QC simulators, <http://www.quantiki.org/wiki/List_of_QC_simulators>
4 QuTIP: the Quantum Toolbox in Python. <http://qutip.org/>
5 Django, a Python web framework: <https://www.djangoproject.com/>

The QuTIP quantum library has a number of dependencies that made it difficult to install at first. In order to get QuTIP working correctly, one must also install scipy, install IPython, install Python dev tools, and install Cython.

Displaying Circuits
QuTIP has a neat feature built in that allows quantum circuits to be rendered as images using LaTeX. Initially, this wasn't working for me: to fix it, one must install pdflatex, pdf2svg, pdfcrop, imagemagick, and xzdec; furthermore, one must also install the LaTeX template "standalone." After installing all of these, QuTIP's quantum circuit rendering works nicely.

Concurrency Bug
It takes a little bit of time for each new gate to be added to the circuit, because QuantumMechanic must re-render each circuit image using LaTeX whenever the user adds a new gate. If the user were to submit another request to the server to add a gate in the meantime, QuantumMechanic would misbehave: the state of the circuit would be inconsistent, or the rendering would be incorrect. To fix this concurrency bug, I placed a short lock on the interface after each gate is added – when the server finishes rendering, the lock is released and the user can add another gate.

Checking Circuit Equality
As the QuTIP documentation demonstrates[6], it is possible to export each quantum circuit to a matrix format, allowing us to analyze the circuit as a single unitary transformation. By exporting circuits as matrices, we can compare them to one another, by checking for equality. To avoid floating-point errors, equality checks will be done to within a small margin of error.

Future Steps
The current version is a proof-of-concept, demonstrating the minimal necessary features. As I have more time, there are a variety of features that I would like to add.

Currently, the website is deployed on a laptop in my dorm room; this is not ideal, especially if there are security vulnerabilities in my code that I haven't found yet. In the future, I would like to deploy this more properly, using scripts.mit.edu.

As mentioned, the interface is currently very simplistic: it could be improved upon to use a drag-and-drop interface. This would involve calculating which qubit wire the user wants to add a gate to, based on the coordinates of the point where the user dragged-and-dropped the gate.

6 Decomposition of the Toffoli gate in terms of CNOT and single-qubit rotations. <http://nbviewer.ipython.org/github/qutip/qutip-notebooks/blob/master/examples/example-qip-toffoli-cnot.ipynb>