CS 101-3: Group project

Team selection and choice of project: May 24, 23:59 PST
Project deadline: June 6, 23:59 PST

Quick summary

- Choose your project from the list below (or propose one) by May 24, 23:59 PST. If you want to propose a project, email the instructor and the TA with your idea.
- Teams of 2-3 students (max 3 students per team).
- Each team should have a representative who will turn in the project on Moodle by June 6, 23:59 PST.
- Code + written report.
- Code comprises of your implementation of the project as well as tests/examples.
- Code should also be documented and have a Readme file explaining how to run it on the tests you considered.
- Written report should be a detailed explanation of your implementation, how you tested it, what types of tests you considered and what results you obtained. Pretty plots are strongly encouraged (though not required) :)
- No restriction on programming languages, however you must use at least one *quantum* programming language like Q#, Qiskit, Cirq etc.
- Written report should be at most 10 pages.

Overview and project structure

The group project involves solving a more involved programming assignment pertaining to quantum computation in a team of 2-3 students. A list of suggested projects is provided below, though you can also propose your own project. You are **not** required to use Q# and should feel free to use whichever programming language(s) you like. The goal of the project is to delve deeper into one specific application of quantum algorithms or a specific concept pertaining to quantum computation/programming.

The project should consist of a code component and a written report. You need to choose one of the projects from the list below (or propose your own) and implement a specific application. For instance,

if you choose to implement a quantum algorithm (like Kuperberg's algorithm) then your project should consist of an implementation of that algorithm in a quantum programming language as well as a report on your implementation. You need to "demonstrate" the correctness of your implementation. This is done in two ways. First, you should have a number of examples or tests for your project, illustrating that your implementation works and that the results obtained are what we'd expect them to be. Second, you need to write a report of up to 10 pages describing the project, how you implemented it as well as how you tested your implementation.

Grading criteria

The grading scheme is the following:

- **60% Coding part**, broken down as follows:
  - *35% Core implementation.* Most of the points will be for the correctness of your implementation.
  - *15% Testing.* These are points allotted for the tests you considered in your application. You should try to make these tests as comprehensive as possible. The general idea with the tests is to convince yourselves, as well as anyone who might look at your implementation, that it is working correctly. For these tests you should specify what the output you obtained was and what the expected output should be.
  - *10% Code documentation.* These are points allotted for how well your code is documented. This primarily refers to comments in your code as well as a Readme file describing how to run it (say on the tests you considered). You don't need to adhere to a specific coding style, though you should aim to make your implementation easily readable.
- **40% Written part**, broken down as follows:
  - *10% General project description.* Which project did you choose? What is the main problem that needs to be solved? What is the proposed solution? How did you implement that solution?
  - *20% Detailed description of your implementation and results.* What is the logic behind your implementation? How did you make use of the (quantum) programming language? If you chose a language that wasn't covered in the course, why did you opt for that particular language? What results did you obtain? Do they match the expected results?
  - *10% Description of the tests.* What kind of tests did you consider? How did you choose them? How many tests did you consider? Why?

Suggested projects

The projects are divided into projects pertaining to classical simulation of quantum computation, projects that involve quantum algorithms or protocols and speculative/research-ish projects. For each suggested project (except the speculative ones) you are provided one or two papers that contain a detailed description of the application you have to implement. You are free (and encouraged), however, to seek out other resources as well.

Classical simulation projects

The goal with these projects is to simulate quantum circuits classically. As such, for these projects you will not be using a quantum programming language for the core coding component. Instead, you should use the quantum programming language to compare your quantum circuit simulator with that of the language. For instance you can simulate circuits with your simulator and compare the results to running those circuits in Q#.

1. **Simulating circuits dominated by Clifford gates**. Implement a simulator of quantum circuits whose runtime is exponential in the number of T gates of the circuit (rather than exponential in the total number of qubits or the total number of gates). This simulator should be efficient for circuits having a small number of T gates (Clifford gates are H, $CNOT$, $S = \sqrt{T}$ and any combination of these gates). See: `https://arxiv.org/abs/1601.07601`.

2. **Simulating quantum computation by contracting tensor networks**. Some of the most efficient methods for simulating quantum circuits involve contracting tensor networks. Implement a quantum circuit simulator using tensor network contraction. See: `https://arxiv.org/abs/quant-ph/0511069`.

3. **Simulating noisy IQP circuits in polynomial time**. In Homework 4 you were introduced to IQP circuits as another way of obtaining quantum computational supremacy-type problems. The linked paper gives a polynomial-time classical algorithm for simulating noisy IQP circuits. Implement it. See: `https://arxiv.org/pdf/1610.01808.pdf`.

Quantum algorithms and protocols

These projects involve implementing a quantum algorithm or protocol, hence the core coding part must use a quantum programming language.

1. **Kuperberg's algorithm for the dihedral coset problem**. Simon's and Shor's algorithms are instances of the *abelian hidden subgroup problem* that quantum computers can solve efficiently. A question of major importance (especially pertaining to the security of cryptographic protocols) is whether there are efficient quantum algorithms for the non-abelian version. Kuperberg's algorithm is a sub-exponential time quantum algorithm the hidden subgroup problem over the dihedral group (which is non-abelian). Implement it. See: `https://arxiv.org/abs/quant-ph/0302112`.

2. **Quantum algorithm for semidefinite programming (SDP)**. Semidefinite programming is a convex optimization problem. Recently, a number of efficient quantum algorithms have been proposed for SDP. Implement one such algorithm. See:

   - https://arxiv.org/abs/1909.04613
   - https://arxiv.org/abs/1710.02581

3. **Quantum algorithms for partial differential equations (PDEs)**. Implement a quantum algorithm for solving PDEs. See: `https://arxiv.org/abs/2002.07868`.

4. **Schumacher compression**. Implement the Schumacher compression scheme for compressing qubits. See:

   - https://arxiv.org/pdf/quant-ph/9603009.pdf
   - https://www.dias.ie/wp-content/uploads/2012/06/lecture-4-wilde.pdf

5. **Quantum Lovász Local Lemma algorithm**. QLLL is an algorithm that can solve certain instances of QSAT (quantum satisfiability problem), which asks whether a local Hamiltonian has ground-state energy zero. Implement this algorithm. See: `https://arxiv.org/pdf/1112.1413.pdf`.

6. **Quantum algorithm for learning Disjunctive Normal Form (DNF) formulas**. Given black box access to a a boolean formula in DNF form, is it possible to learn what the DNF is efficiently? Quantumly, yes. Implement this algorithm: `http://mathcs.duq.edu/~jackson/quantex.pdf`.

7. **Quantum reinforcement learning**. Implement a quantum algorithm for reinforcement learning and compare its performance to a classical algorithm, such as Q-learning. See:

   - `https://arxiv.org/pdf/0810.3828.pdf`
   - `https://arxiv.org/pdf/1811.08676.pdf`
   - `https://en.wikipedia.org/wiki/Q-learning`

8. **Quantum recommendation systems**. Implement the quantum algorithm for recommendation systems. This algorithm was one of the first conjectured examples of a quantum machine learning algorithm with a clear advantage over existing classical algorithms. However, the algorithm was "dequantized" in a breakthrough result of Ewin Tang. Still, the quantum version still has better asymptotic scaling than the dequantized algorithm. See:

   - `https://arxiv.org/abs/1603.08675`
   - `https://arxiv.org/abs/1807.04271`

9. **Approximating the Jones Polynomial**. An interesting problem that was shown to be BQP-complete (i.e. as hard as any other problem that can be solved efficiently by quantum computers) comes from quantum field theory and involves approximating a polynomial known as Jones Polynomial. Implement the algorithm performing this approximation. See: `https://arxiv.org/abs/quant-ph/0511096`.

10. **Simulating quantum field theories on a quantum computer**. On the subject of quantum field theory (QFT), certain QFTs can be simulated on a quantum computer. Implement such a quantum simulation. See:

    - `https://arxiv.org/abs/1111.3633`
    - `https://arxiv.org/abs/1811.10085`

11. **Fast Hamiltonian simulation**. In the lectures we discussed the Trotter-Suzuki method for quantum simulation. However, better methods exist. Implement one such method, for instance the one from this paper based on randomized compiling: `https://arxiv.org/abs/1811.08017`.

12. **Determining electrical resistance in a network**. An important class of quantum algorithms that we did not cover in the course is that of quantum walks. One nice application of them is an efficient quantum algorithm for determining the electrical resistance in networks. Implement that algorithm. See: `https://arxiv.org/abs/1311.1851`.

Speculative ideas

These projects are more research-oriented, in the sense that there isn't a very well-defined result. The goal is for you to explore different things and see whether you arrive at interesting results.

1. **Comparing classical and quantum machine learning**. The point of this project is to see for instance whether quantum classifiers (like the one from Homework 4) learn faster than small classical neural networks. Compare small quantum neural networks with small classical networks and see whether the quantum case provides an advantage. You should try to make it so that the classical neural network is comparable in size to the quantum circuit you're running. See:

   - `https://arxiv.org/abs/1408.7005`
   - `https://arxiv.org/abs/1804.00633`
   - `https://arxiv.org/abs/1710.01022`
   - `https://arxiv.org/abs/1509.04279`

2. **Finding interesting patterns in outputs of quantum computational supremacy circuits**. We've seen in the lectures that random quantum circuits have special properties (like the Porter-Thomas distribution of the output probabilities, anti-concentration etc). Are there other interesting properties that we've missed? The goal of this project is to try to determine this. Use, for instance, unsupervised machine learning to discover patterns/properties of families of quantum circuits. You should then see whether these patterns exist for quantum circuits that have only single-qubit gates (and hence do not create entanglement) or in random classical circuits under various input distributions. Essentially the point is to give evidence that the found patterns are specific to quantum computational supremacy circuits. See:

   - `https://arxiv.org/abs/1809.07442`
   - `https://arxiv.org/abs/1608.00263`
   - `https://arxiv.org/abs/1803.04402`

A good resource if you want to propose your own project is to have a look at the quantum algorithms zoo: `http://quantumalgorithmzoo.org/`.