CS269Q: Quantum Programming **Spring 2019**

Final Project Guidelines

May 9, 2019

The course project is the capstone of the class. *It is preferred that you do these projects in groups of three, however groups of less than three will accepted.* There are two ways to approach the project:

- **(Programming Project)** You and your group will choose a topic in quantum computing and implement an interesting program along with a short report (3-5 pages) that is relevant to that topic.
- **(Theory Project)** You and your group will choose a topic, research a few relevant papers, and write a research report 7-10 pages that covers the theory results for your project.

In both approaches, original research is encouraged, but not necessary. A list of suggested project topics is provided below. You and your group can either choose from these topics or suggest an alternative of your own design. This is an opportunity to combine your interests and your own background (in machine learning, optimization, distributed systems, physics, etc) with what you've learned in this course.

Importantly, you will need to pick a project that is doable in the time frame given (from mid-May to June 6th). You and your group will have to submit a project proposal by May 17th, and I will give guidance on whether the scope is appropriate for the course.

Here are some suggestions on how you can pick topics for your project:

1. Choose a suggested project topic area from the list at the end of this document.
2. Take something youre already doing research in, and explore if there is a quantum version of it.
3. Take a topic covered in the class, and investigate it more deeply.
4. Take a topic in quantum computing you've always wanted to learn about (but wasn't talked about in class).

1 Deliverables

Here is what is expected of you (the last one is optional and the middle ones depend on if you are doing a programming or a theory project):

1. **(Project Proposal)** Your group will submit a proposal on Gradescope that includes your group names. This proposal is due on **May 17th**. This should be at most one page describing your proposed project, along with some relevant papers/resources. A strong initial proposal will include the topic area, the general structure of the intended implementation, as well as the key results that will be intended to use in the research report. I will give feedback and suggest additional papers or implementation suggestions. Feel free to make an appointment to discuss your project in person.

2. **(Project Progress Update)** A less than one page written update on your project progress from each group. This is due on **May 27th**. It should describe the progress you have made against your original proposal, including any changes of direction and results so far.

3. **(Programming Project Only: Code Repository)** No later than **June 6th** your group should upload your final project code to either (i) a publicly viewable repository (github, gitlab, bitbucket, etc.) or (ii) a private repository whose access credentials are provided in your report (see below). Your code should include full instructions on how to install your project as well as instructions for how to get started and use some of its key features. If you have alternative packaging methods that are needed please include these in your project proposal.

4. **(Programming Project Only: Short Written Report)** Your group must upload a final report no later that **June 6th** to Gradescope. It should be 3-5 pages long. It can be longer, but we will only read submissions longer than 5 pages at our own discretion. Please use reasonable margins and font sizes. This report will describe what is interesting about your project and the data/examples you have taken with your implementation to show that it works. It should also include some remarks on potential future extensions of your work. Your report should describe how to find and access the repository with your code.

5. **(Theory Project Only: Written Report)** Your group must upload a final report no later that **June 6th** to Gradescope. It should be 7-10 pages long. It can be longer, but we will only read submissions longer than 10 pages at our own discretion. Please use reasonable margins and font sizes. This report will describe the research background for your project and what extensions / constructions / proofs you have made in your project. It should also include some remarks on potential future extensions of your work.

6. **(Online Showcase)** We plan to showcase the students project on the course homepage for other students and researchers to see what cool things you've done. Participation in the showcase is optional, and has no effect on your grade. Benefits: your project gets some publicity, and also the quantum computing community benefits from what you've built! You have the option of making either/both/none of the report and code accessible openly.

2 Types of Projects

Here are some examples of types of projects that you can do:

1. **Compare implementations across frameworks.** In these projects you can take an example algorithm or compilation trick and implement it in a few different frameworks. In the course we have mostly used pyQuil, but there's also IBM Qiskit, Microsoft Q, Project Q, Pennylane and Strawberry Fields and others. For a list of frameworks to potentially work in check out: `https://qosf.org/project_list/`

2. **Implement a small instance on a real QPU.** Here you will take an algorithm that runs on a simulator and massage it into a small instance that runs on a QPU. This can be tough! Today's QPUs are small and very noisy. However, this type of project is likely to be very rewarding and may even end up as a potential publication after further development and work. For this course you will have access to Rigetti's QCS system and IBM also provides a small QPU backend with its IBMQ platform. Should you wish to use the Rigetti QCS system please indicate this in your proposal and we'll get you set up.

3. **Benchmark implementations across emulators.** Some quantum algorithms (Shor's, Grover's, HHL) are too big to be run or even simulated today. In those cases you can write implementations that compile into programs, but that these programs can't yet be run except in very small instances. You can instead compare different methods of implementing these algorithms and compare program outputs along metrics like of qubits or gate depth.

4. **Build developer tools.** In the following course project areas there are lots of developer tool suggestions. A project of this type would show the main feature(s) of the tool and how it helps with quantum programming in the near or far.

5. **Theory Project**: Extend an algorithm from the `https://quantumalgorithmzoo.org/`

3 Course Project Topics

Here are some potential course project topics. This is by no means an exhaustive list of things you can do. If you end up curious about an area and are looking for more references then please do ask on Piazza and we'll be happy to follow up. If you are struggling with choosing a project then we recommend you choose the Default project (given below). You can also schedule an appointment to discuss your interests and brainstorm a project.

1. **Default Project** Write your own version of a QVM in a language of your choosing (default is Python). This QVM should accept as many legal Quil statements as possible. The grammar for Quil is given here `http://bit.ly/quil-spec` with semantics described by Smith et al. [2016]. Benchmark the performance of your QVM against the Rigetti QVM.

2. Pick a theoretical algorithm from `https://quantumalgorithmzoo.org/`. Implement a version of the algorithm in pyQuil. What are concrete runtimes and memory needs (Quil instruction depths and qubit numbers) for specific small instances?

3. Single qubit quantum state tomography comparisons. There are several ways to reconstruct a quantum state from tomographic analysis. Several examples are given in Schmied [2014]. Implement and compare a few of these methods and use them to perform tomography on qubits on a real QPU.

4. Implement and benchmark a simulated version of Variational Quantum Unsampling Carolan et al. [2019], a new technique for analyzing quantum programs.

5. Pick a language that has no support for quantum programming. Build a pyQuil like library to generate Quil using this language. What new features can you now use?

6. OpenQASM is an intermediate representation for quantum programming that is similar to Quil. Write a transpiler that takes Quil code and converts it into OpenQASM for as many language features as possible. For more details on OpenQASM see Cross et al. [2017] and the online spec at `https://github.com/Qiskit/openqasm`.

7. Implement and benchmark the traveling salesman problem using QAOA on the QVM.

8. Inserted Tomography for breakpoint debugging. In general one doesn't have access to the wavefunction of the quantum memory of a real QPU. This can make debugging difficult as we want to look at the wavefunction of the quantum memory at different steps in our program. Write a small pyQuil library that makes it easy to insert breakpoints into a pyQuil program that will run tomography at that breakpoint and reconstruct the wavefunction of the quantum memory at that step in the program. Note that this will only work on subsets of qubits, so perhaps an argument of the breakpoint is what qubits to do the tomography on.

9. Benchmarking Optimizers. Pick a variational quantum problem, e.g. QAOA with MAXCUT, and test several ways of optimizing the variational angles in the program. What classical methods of optimization work best? Why?

10. Simulating the Ising Model with VQE. In Cervera-Lierta [2018] an example is given of a simulation of a physical system called a 1D Ising model. Make your own implementation and benchmark of this simulation.

11. Choose a constrained optimization problem that has a reduction to MAX-CUT. Use this reduction (and any tricks you can find) to make a QAOA implementation of this algorithm that runs on the QVM or could run on a QPU. Benchmark this implementation.

12. In Gottesman [2016] several examples are given for small quantum circuits (5 qubits or less) that benchmark fault tolerant operations. Implement these circuits and test them on a noisy QVM and/or a QPU.

13. The Steane code from Steane [1996] improves upon the Shor code as a small code for quantum error correction. Implement this code for a single qubit and plot the logical error rate vs. different physical error rates under different noise models for this code. You could also do this for any other quantum error correcting code such as those described in Devitt et al. [2009].

14. In Abramsky and Brandenburger [2011] a family of inequalities are described that provide a metric for contextuality in a system. Use this framework to test the contextuality of states prepared on a noisy QVM and/or QPU.

15. Compare how different choices of hardware ansatzes perform on a range of VQE problem instances. Why should some work better over others? If some qubits are much noisier than others, how does this affect what ansatz you should choose?

16. Given a quantum program and a quantum instruction set architecture (a topology along with noise characterizations for your QPU) write a compiler that converts that program to one that is better optimized for your particular quantum ISA. How much better can you do?

17. Implement and test a quantum secret sharing scheme protocol as described in Hillery et al. [1998] and Cleve et al. [1999].

18. Write a tutorial - with interactive notebooks and that is accesible to students with background from this class - that shows how to use OpenFermion to simulate electronic structure problems on quantum computers.

19. Write a simulation of the Surface Code quantum error correction protocol Fowler et al. [2012]. Use it to calculate a threshold for the surface code.

20. Color codes Landahl et al. [2011] are another version of a topological quantum error correcting code that is different from the surface code. Implement a color code on some small lattices and calculate the fault-tolerant threshold.

21. (Theory Project) What is the analog for Shannon channel capacities for quantum channels? How can we use this capacity to show optimality bounds for quantum error correction or quantum algorithms?

22. (Theory Project) Investigate Kuperberg's algorithm for the dihedral coset problem (DCP) Kuperberg [2003]Kuperberg [2011]. What is the concrete running time analysis of the algorithm? The papers only gives an asymptotic analysis.

23. (Theory Project): Extend an algorithm from the `https://quantumalgorithmzoo.org/`. Pick an instance (or family of instances) of the algorithms where you can calculate concrete runtimes analytically.

4 Important Dates

- **May 17th**: Project proposals due.
- **May 27th**: One page project update due.
- **June 6th**: Code repository and short report due.

References

Robert S. Smith, Michael J. Curtis, and William J. Zeng. A Practical Quantum Instruction Set Architecture. 8 2016. URL `http://arxiv.org/abs/1608.03355`.

Roman Schmied. Quantum State Tomography of a Single Qubit: Comparison of Methods. 7 2014. doi: 10.1080/09500340.2016.1142018. URL `http://arxiv.org/abs/1407.4759http://dx.doi.org/10.1080/09500340.2016.1142018`.

Jacques Carolan, Masoud Mosheni, Jonathan P. Olson, Mihika Prabhu, Changchen Chen, Darius Bunandar, Nicholas C. Harris, Franco N. C. Wong, Michael Hochberg, Seth Lloyd, and Dirk Englund. Variational Quantum Unsampling on a Quantum Photonic Processor. 4 2019. URL `http://arxiv.org/abs/1904.10463`.

Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open Quantum Assembly Language. 7 2017. URL `http://arxiv.org/abs/1707.03429`.

Alba Cervera-Lierta. Exact Ising model simulation on a quantum computer. 7 2018. doi: 10.22331/q-2018-12-21-114. URL `http://arxiv.org/abs/1807.07112http://dx.doi.org/10.22331/q-2018-12-21-114`.

Daniel Gottesman. Quantum fault tolerance in small experiments. 10 2016. URL `http://arxiv.org/abs/1610.03507`.

Andrew Steane. Multiple Particle Interference and Quantum Error Correction. 1 1996. doi: 10.1098/rspa.1996.0136. URL `http://arxiv.org/abs/quant-ph/9601029http://dx.doi.org/10.1098/rspa.1996.0136`.

Simon J. Devitt, Kae Nemoto, and William J. Munro. Quantum Error Correction for Beginners. 5 2009. doi: 10.1088/0034-4885/76/7/076001. URL `http://arxiv.org/abs/0905.2794http://dx.doi.org/10.1088/0034-4885/76/7/076001`.

Samson Abramsky and Adam Brandenburger. The sheaf-theoretic structure of non-locality and contextuality. *New Journal of Physics*, 2011. ISSN 13672630. doi: 10.1088/1367-2630/13/11/113036.

M. Hillery, V. Buzek, and A. Berthiaume. Quantum secret sharing. 6 1998. doi: 10.1103/PhysRevA.59.1829. URL `http://arxiv.org/abs/quant-ph/9806063http://dx.doi.org/10.1103/PhysRevA.59.1829`.

Richard Cleve, Daniel Gottesman, and Hoi-Kwong Lo. How to share a quantum secret. 1 1999. doi: 10.1103/PhysRevLett.83.648. URL `http://arxiv.org/abs/quant-ph/9901025http://dx.doi.org/10.1103/PhysRevLett.83.648`.

Austin G. Fowler, Matteo Mariantoni, John M. Martinis, and Andrew N. Cleland. Surface codes: Towards practical large-scale quantum computation. 8 2012. doi: 10.1103/PhysRevA.86.032324. URL `http://arxiv.org/abs/1208.0928http://dx.doi.org/10.1103/PhysRevA.86.032324`.

Andrew J. Landahl, Jonas T. Anderson, and Patrick R. Rice. Fault-tolerant quantum computing with color codes. 8 2011. URL `http://arxiv.org/abs/1108.5738`.

Greg Kuperberg. A subexponential-time quantum algorithm for the dihedral hidden subgroup problem. 2 2003. URL `http://arxiv.org/abs/quant-ph/0302112`.

Greg Kuperberg. Another subexponential-time quantum algorithm for the dihedral hidden subgroup problem. 12 2011. URL `http://arxiv.org/abs/1112.3333`.

Thanks to Henry Yuen and his *Quantum Computing: Foundations to Frontier* for inspiring some of the design of this final project. `http://www.henryyuen.net/classes/fall2018/`