AQI: Advanced Quantum Information
Lecture 3 (Module 4): Simulation, runtimes and complexity

February 21, 2013

Lecturer: Dr. Mark Tame
(email: m.tame@imperial.ac.uk)

Introduction

One of the big applications of classical computing is the simulation of physical systems. Here, simulation can be thought of as a kind of algorithm. It turns out that not all physical systems can be simulated efficiently on a classical computer. In particular, the simulation of quantum systems is possible, but in general very inefficient. For instance, in the last 30 years quite a lot of progress has been made using Monte Carlo methods and Density Matrix Renormalisation Group techniques. Unfortunately, these techniques are limited in what systems they cover and the physical properties that can be extracted due to the approximations made.

With a quantum computer at our disposal, however, we can simulate a much larger class of quantum systems. Why is this important? It turns out that there are a number of interesting potential applications. My favourite is the potential to efficiently simulate and test new types of quantum materials, *i.e.* those with macroscopic behaviour that emerges out of complex microscopic quantum dynamics. Such simulations may shed light on new properties of materials such as graphene and other condensed matter systems that are too challenging to simulate with classical methods.

1 Quantum simulation

1.1 What's the problem?

Take a simple example using the Schrödinger equation

$$i\hbar\frac{d}{dt}|\psi\rangle = H|\psi\rangle . \tag{1}$$

For a single qubit we can turn this into a set of coupled differential equations

$$\begin{aligned}
\langle 0| i\hbar\frac{d}{dt}|\psi\rangle &= \langle 0| H\Big(|0\rangle\langle 0| + |1\rangle\langle 1|\Big)|\psi\rangle \\
\to i\hbar\frac{d\langle 0|\psi\rangle}{dt} &= \langle 0|H|0\rangle\langle 0|\psi\rangle + \langle 0|H|1\rangle\langle 1|\psi\rangle \\
\to i\hbar\frac{d\langle 1|\psi\rangle}{dt} &= \langle 1|H|0\rangle\langle 0|\psi\rangle + \langle 1|H|1\rangle\langle 1|\psi\rangle
\end{aligned}$$

$$\to i\hbar\frac{d\psi_0(t)}{dt} = H_{00}\psi_0(t) + H_{01}\psi_1(t) \tag{2}$$

$$\to i\hbar\frac{d\psi_1(t)}{dt} = H_{11}\psi_1(t) + H_{10}\psi_0(t) \tag{3}$$

For example, $H = Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$ gives $H_{00} = 1,\ H_{01} = 0,\ H_{10} = 0,\ H_{11} = -1$. Or we could have $H = X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$, which gives $H_{00} = 0,\ H_{01} = 1,\ H_{10} = 1,\ H_{11} = 0$.

Pretty simple, huh? However, for 2 qubits we have 4 coupled equations and if the qubits are interacting then H becomes more complex. For n qubits we have 2^n coupled equations, which corresponds to an exponential growth in the number of differential equations!

Sometimes we can make approximations, but in general this isn't the case and therefore we end up with an exponential growth in the runtime of the simulation. Another way to see this is the following. For the solution to Eq. (1) we have

$$|\psi(t)\rangle = e^{-\frac{i}{\hbar}Ht}\,|\psi(0)\rangle\,, \tag{4}$$

where the left hand side is a vector of dimension 2^n, the exponential is a unitary matrix of dimension $2^n \times 2^n$ and the state on the right is a vector of dimension 2^n. This means that $2^n \times 2^n = 2^{2n}$ operations are required and a memory space of $\sim 2^{2n}$.

While a quantum computer allows us to reduce the memory space, we still need to exponentiate H (which can be extremely difficult as its size grows exponentially) and find an efficient way to implement the resulting U using a quantum circuit with logic gates.

1.2 Efficient approximation to the problem

If $H = \sum_{k=1}^{L} H_k$, where H_k acts on at most a constant c number of systems, then this implies that L is polynomial in n, where n is the number of particles. The particles don't necessarily have to be spin 1/2's (qubits) and can be qudits (which we can later decompose into qubits). For example, if the H_k are single-body (X_i) and two-body (X_iX_j) interactions, as is the case in Hubbard and Ising models, then it turns out that they can be exponentiated efficiently as $e^{-\frac{i}{\hbar}H_k t}$ and an efficient circuit can be found using polynomial operations and gates. However, the total evolution is harder to calculate as $e^{-\frac{i}{\hbar}Ht} \neq \Pi_k e^{-\frac{i}{\hbar}H_k t}$, because $[H_i, H_j] \neq 0$ in general. Fortunately we can use an asymptotic approximation called the 'Trotter formula'

$$\lim_{m\to\infty}\left(e^{iAt/m}e^{iBt/m}\right)^m = e^{i(A+B)t} \qquad \text{where } \frac{t}{m} = \Delta t. \tag{5}$$

Here, A and B are any Hermitian operators and $[A, B] = 0$ doesn't necessarily have to be true. However, as we can only use a finite number of steps 'm' in our simulation we'll get some error term. It can be shown using the Baker-Campbell-Hausdorf formula that

$$e^{i(A+B)\Delta t} = e^{iA\frac{\Delta t}{2}}e^{iB\Delta t}e^{iA\frac{\Delta t}{2}} + \mathcal{O}(\Delta t^3) \tag{6}$$

so that

$$U_{\Delta t} = \Pi_{k=1}^{L} e^{-\frac{i}{\hbar}H_k\Delta t}\Pi_{k=L}^{1} e^{-\frac{i}{\hbar}H_k\Delta t} \tag{7}$$

leads to

$$U_{\Delta t} = e^{-\frac{i}{\hbar}H(2\Delta t)} + \mathcal{O}(\Delta t^3) \tag{8}$$

which gives

$$U = e^{-\frac{i}{\hbar}H(2\Delta t)} = U_{\Delta t} + \mathcal{O}(\Delta t^3) \tag{9}$$

and therefore

$$U_{\Delta t} \simeq e^{-i\frac{i}{\hbar}H(2\Delta t)} \tag{10}$$

up to some error term which we can make arbitrarily small by choosing the number of steps m and time step Δt (as shown below). In Eq. (10) the left hand side is the approximation to U that we use for our simulation. It can be found and implemented efficiently. On the right hand side we have the true unitary evolution of the system. This is hard to find and implement efficiently.

So in summary, using a quantum state for $|\psi\rangle$ gives us efficient memory space and by using a unitary operation $U_{\Delta t}$ gives us an efficient runtime for our simulation.

$$\begin{array}{rclll} U\,|\psi_0\rangle & = & |\psi\rangle & \text{for time } t & \text{system} \\ U_{\Delta t}^m\,|\psi_0\rangle & = & \left|\tilde{\psi}\right\rangle & \text{for time } 2\Delta t m = t & \text{simulation} \end{array}$$

Here, L is polynomial in n, the number of particles, and m is proportional to L (see Eq. (7)). So m is a runtime that's polynomial in n.

One can show that the error between the two operations U and $U_{\Delta t}^m$ is bounded

$$E(U_{\Delta t}^m, e^{-\frac{i}{\hbar}Ht}) \leq m\alpha\Delta t^3 \qquad \text{for some constant } \alpha. \tag{11}$$

Here, E is the error between measurements on $|\psi\rangle$ and $\left|\tilde{\psi}\right\rangle$, and α depends on the measurement. So for keeping a fixed error E when we increase the number of steps m, we must decrease the step size Δt. But the bound on E grows as Δt^3, so t can grow with linear overhead in m, keeping E fixed.

The above insight is quite a promising result for simulating quantum systems on a quantum computer. What's perhaps even more interesting is that it's still an open question as to what systems can and can't be simulated efficiently on classical and quantum computers. For instance, it's known that unitary operations exist that can't be approximated efficiently even on a quantum computer!

2 Runtimes

So far in this module I've talked about hard/easy problems and efficient/not efficient operations, but I should really quantify what I mean. To help with this we can use asymptotic bounds. These help characterise an algorithm's efficiency and allow us to compare the performance of two algorithms doing the same task. For large values of inputs/components the multiplicative constants and lower order terms of an exact running time are dominated by the effects of the input size (bits/qubits) and the number of components (logic operations).

2.1 Asymptotic tight bound Θ

A function $f(n) \in \Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ $\forall n \geq n_0$. An example of this is given in the figure below:

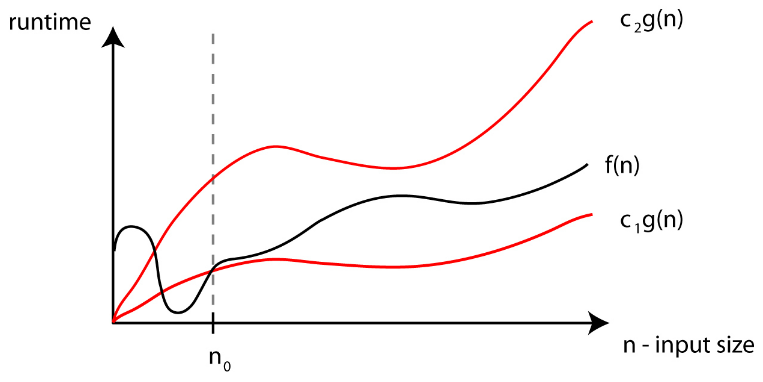

Figure 1: Asymptotic tight bound.

Note a slight abuse of notation (that many people do and is ok) is to write $f(n) = \Theta(g(n))$.

Here's an example:

$$\frac{1}{2}n^2 - 3n = \Theta(n^2) \tag{12}$$

which means that

$$\begin{aligned} c_1 n^2 &\leq \frac{1}{2}n^2 - 3n \leq c_2 n^2 \\ &\rightarrow c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2 \end{aligned} \tag{13}$$

This inequality is satisfied by $c_1 \leq \frac{1}{14}$ for $n \geq 7$ and $c_2 \geq \frac{1}{2}$ for $n \geq 1$. Thus $\frac{1}{2}n^2 - 3n = \Theta(n^2)$ for $n \geq 7$.

2.2 Asymptotic upper bound $\mathcal{O}$

A function $f(n) \in \mathcal{O}(g(n))$ if there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)\ \forall n \geq n_0$. An example of this is given in the figure below:

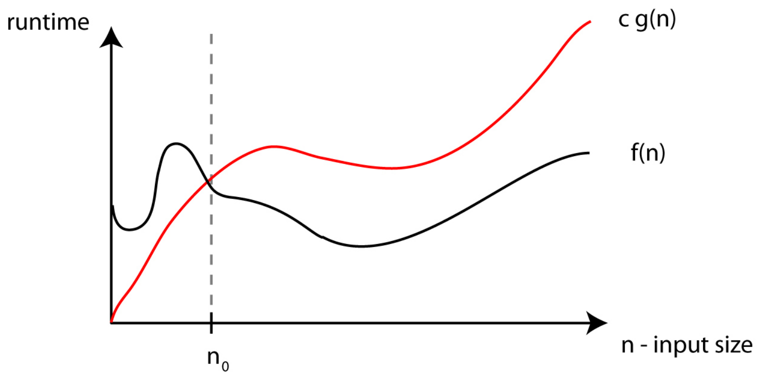

Figure 2: Asymptotic upper bound.

Note a slight abuse of notation is to write $f(n) = \mathcal{O}(g(n))$. The upper bound can also be thought of as the 'worst case scenario' for an algorithm.

2.3 Asymptotic lower bound Ω

A function $f(n) \in \Omega(g(n))$ if there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)\ \forall n \geq n_0$. An example of this is given in the figure below:

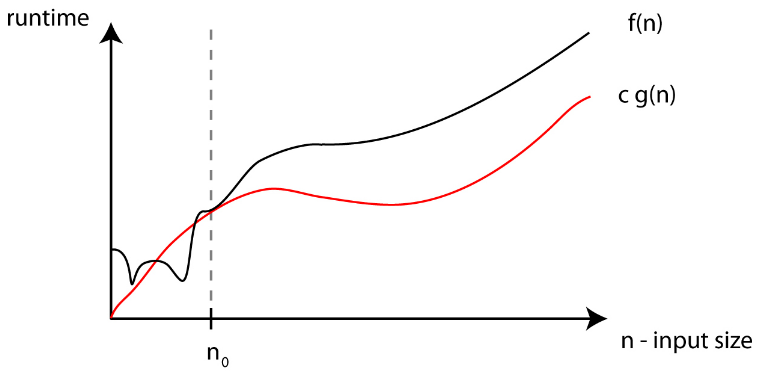

Figure 3: Asymptotic lower bound.

Note a slight abuse of notation is to write $f(n) = \Omega(g(n))$.

One can link all three bounds together by noting that $f(n) = \Theta(g(n))$ iff $f(n) = \mathcal{O}(g(n))$ and $f(n) = \Omega(g(n))$.

3 Complexity classes

In algorithm design we look at the worst case scenario, *i.e.* the upper bound $\mathcal{O}$. Computational complexity on the other hand is all about proving lower bounds Ω for the best possible algorithm for solving a problem, even if we don't yet know what that algorithm is. Computational complexity is complementary to algorithm design in that the most efficient algorithm we could design, with upper bound $\mathcal{O}$, would match perfectly with the lower bounds proved by computational complexity.

Computational complexity involves space and time resources. For an n-bit input, any problem that can be solved with resources polynomial in n is said to be easy (or tractable), anything else has resources exponential in n and is said to be hard (or intractable).

Complexity classes are a collection of problems, all of which share a common feature with respect to the computational resources needed to solve problems. There are many classes, so I won't mention all of them, just some of the important ones. The following figure helps:

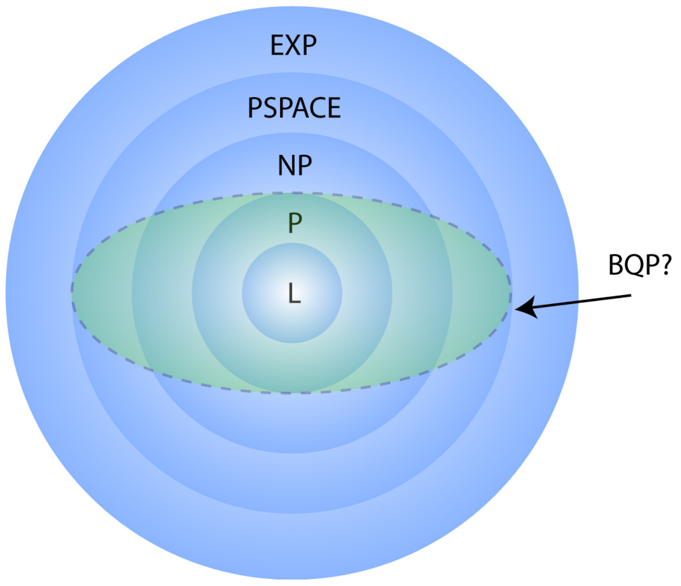

Figure 4: Major complexity classes. Here, BQP is the class of all problems that can be solved efficiently on a quantum computer where a bounded error of probability is allowed.

- L (space): Problems that can be decided by a computer running in logarithmic space. Note, problems are formally formulated as 'decision' problems in computational complexity (decided $\leftrightarrow$ solved).
- P (time): Problems that can be decided in polynomial time.
- NP (time): Problems whose solutions can be checked in polynomial time. Factoring is believed to be in this class. NP-complete is a subclass of NP that consists of the hardest problems in NP. Any algorithm that can solve a specific NP-complete problem can be adapted to solve any other problem in NP with small overhead.

- PSPACE (space): Problems that can be decided using resources polynomial in spatial size, but not necessarily in time.
- EXP (time): Problems that can be decided in exponential time.

Two types of hierarchy allow us to mix space and time resources: $\mathsf{SPACE(\,)}$ and $\mathsf{TIME(\,)}$. These lead to

$$\mathsf{L} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXP} \tag{14}$$

It's known that $\mathsf{P} \subset \mathsf{EXP}$ and $\mathsf{L} \subset \mathsf{PSPACE}$, so one of the $\subseteq$ in Eq. (14) must be a strict subset, but we don't know which one!

We don't know if $\mathsf{P} = \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{NP}$. This is known as the '$\mathsf{P}$ vs NP problem'. Despite decades of work and countless claims, we still don't know! There's a $1 million prize if you can solve it...

References

- S. Lloyd, Science **273** 1073 (1996).
- I. Buluta and F. Nori, Science **326** 5949 (2009).
- Nature Physics Insight into Quantum Simulation (3 review articles): http://www.nature.com/nphys/insight/quantum-simulation (2012).
- T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *Introduction to Algorithms*, MIT press (2001).
- The quantum complexity zoo: https://complexityzoo.uwaterloo.ca/Complexity_Zoo.
- Clay Mathematics Institute Millenium Prize problem for P vs NP: http://www.claymath.org/millennium
- H. Buhrman, R. Cleve, S. Massar and R. de Wolf, *Quantum Communication Complexity*, Rev. Mod. Phys. **82**, 665 (2010).
- N. D. Mermin, *Quantum Computer Science: An introduction*, Cambridge University Press, Cambridge (2007).
- M. A. Nielsen and I. L. Chuang, *Quantum Computation and Quantum Information*, Cambridge University Press, Cambridge (2000).
- J. Preskill, *Quantum Information lecture notes*, http://www.theory.caltech.edu/people/preskill/ph229/#lecture (2004).