Quantum Annealing

NSF/DOE Quantum Science Summer School

Scott Pakin

8 June 2017

Operated by Los Alamos National Security, LLC for the U.S. Department of Energy's NNSA

LA-UR-17-24530

Outline

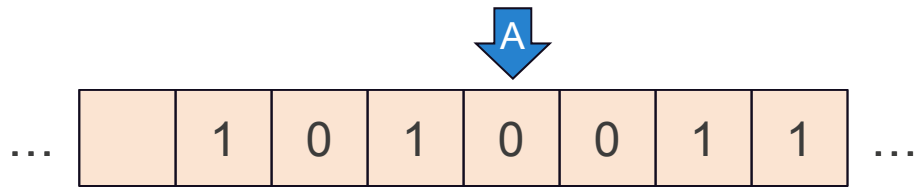
- Performance potential of quantum computing
- Quantum annealing
- Case study: D-Wave quantum annealers
- How to program a quantum annealer
- Parting thoughts

Main Topic to be Addressed

- **What problems can quantum computers solve fast?**
  - What "flavor" of quantum are we referring to?
  - What exactly is a computer?
  - What do we mean by *solve*?
  - What is considered *fast* in this context?

What is a Computer?

- **Mathematical abstraction: a Turing machine**
  - $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$
  - All states, all symbols, blank symbol, input symbols, transition function, initial state, and final states
  - All of the preceding sets are finite, but the memory ("tape") on which they operate is infinite
- **Transition function**
  - Maps {*current state*, *symbol read*} to {*new state*, *symbol to write*, left/right}
  - *Example*: "If you're in state *A* and you see a 0, then write a 1, move to the left, and enter state *B*"



1. *Computing machines.*

We have said that the computable numbers are those whose decimals are calculable by finite means. This requires rather more explicit definition. No real attempt will be made to justify the definitions given until we reach § 9. For the present I shall only say that the justification lies in the fact that the human memory is necessarily limited.

We may compare a man in the process of computing a real number to a machine which is only capable of a finite number of conditions $q_1, q_2, \ldots q_R$ which will be called "m-configurations". The machine is supplied with a "tape" (the analogue of paper) running through it, and divided into sections (called "squares") each capable of bearing a "symbol". At any moment there is just one square, say the r-th, bearing the symbol $\mathfrak{S}(r)$ which is "in the machine". We may call this square the "scanned square". The symbol on the scanned square may be called the "scanned symbol". The "scanned symbol" is the only one of which the machine is, so to speak, "directly aware". However, by altering its m-configuration the machine can effectively remember some of the symbols which it has "seen" (scanned) previously. The possible behaviour of the machine at any moment is determined by the m-configuration q_n and the scanned symbol $\mathfrak{S}(r)$. This pair $q_n, \mathfrak{S}(r)$ will be called the "configuration": thus the configuration determines the possible behaviour of the machine. In some of the configurations in which the scanned square is blank (i.e. bears no symbol) the machine writes down a new symbol on the scanned square: in other configurations it erases the scanned symbol. The machine may also change the square which is being scanned, but only by shifting it one place to right or left. In addition to any of these operations the m-configuration may be changed. Some of the symbols written down

A. M. Turing, "On Computable Numbers, with an Application to the *Entscheidungsproblem*". Proceedings of the London Mathematical Society, 12 November 1936.

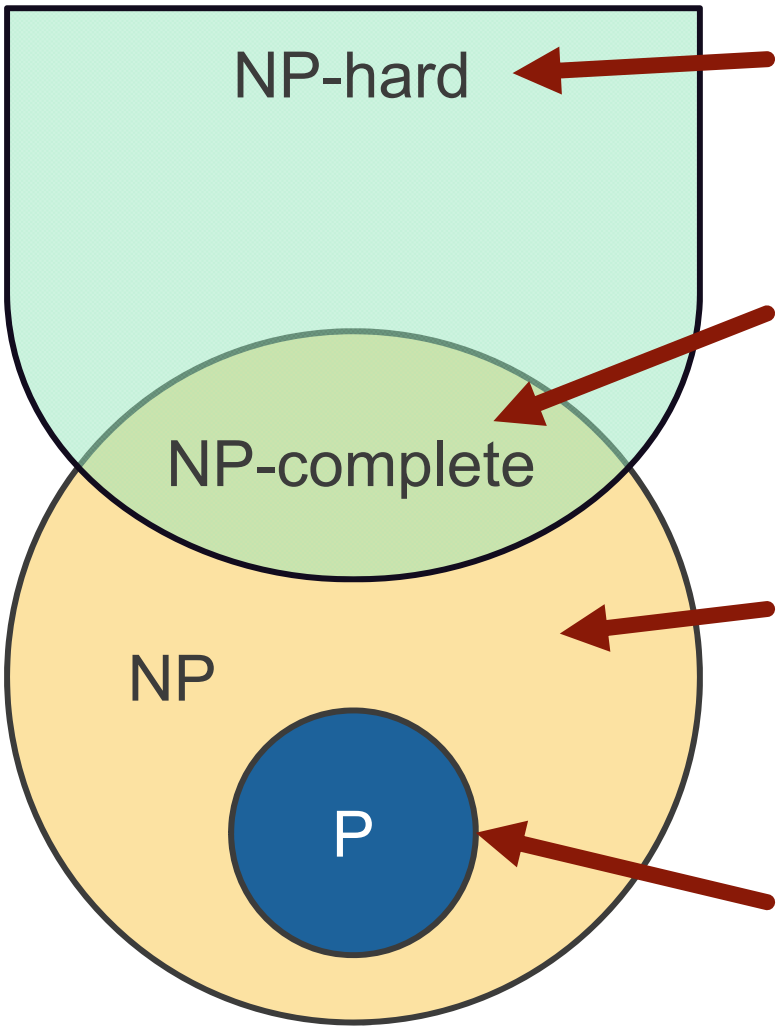
What Else is a Computer?

- **Nondeterministic Turing machine**
  - Replace the transition *function* with a transition *relation*
  - Contradictions are allowed
  - *Example*: "If you're in state *A* and you see a 0, then simultaneously ① write a 1, move to the left, and enter state *B;* ② write a 0, move to the right, and enter state *C*; and ③ write a 1, move to the right, and enter state *B*."
  - At each step, an oracle suggests the best path to take (not realistic, obviously)
- **Quantum Turing machine**
  - Same 7-tuple as in the base Turing machine
  - $M = (Q, \Gamma, b, \Sigma, \delta, q_0, F)$
  - *But…*set of states is a Hilbert space; alphabet is a (different) Hilbert space; blank symbol is a zero vector; transition function is a set of unitary matrices; initial state can be in a superposition of states; final state is a subspace of the Hilbert space
  - No change to input/output symbols; those stay classical

Introduction to Complexity Theory

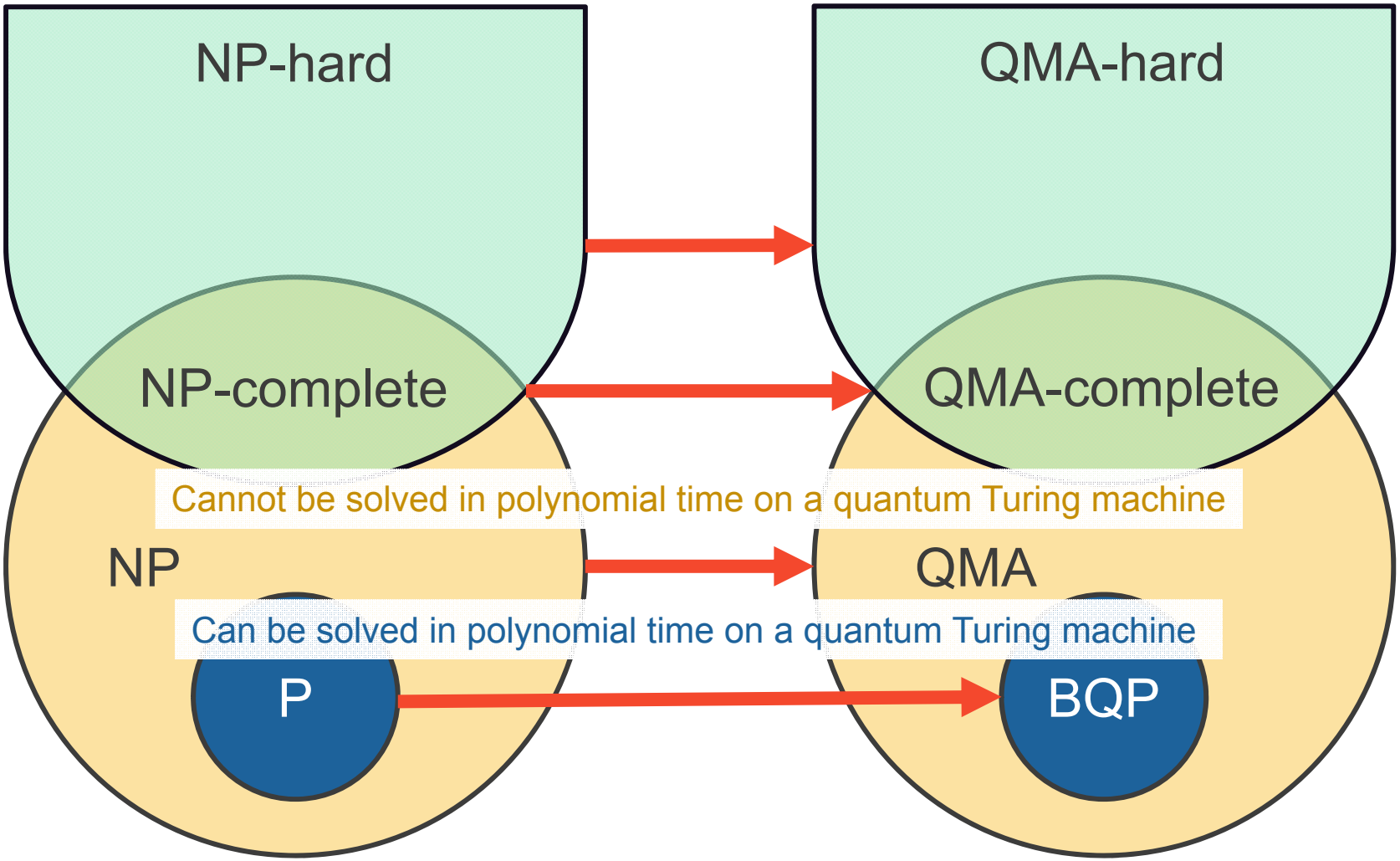
- **What problems can a computer solve quickly?**
- **Discuss in terms of *asymptotic complexity*, not wall-clock time**
  - Ignore constants and all but the leading term
  - For input of size n, $O(n)$ can mean $3n$ seconds or $5n+2 \log n+3/n+20$ hours; it doesn't matter
  - Polynomial time, $O(n^k)$ for any k, is considered good (efficiently solvable), even if an input of size n takes $1000n^{20}$ years to complete
  - Superpolynomial time—most commonly exponential time, $O(k^n)$ for $k>1$—is considered bad (intractable), even if an input of size n completes in only 2^n femtoseconds
- **Categorize problems into complexity classes**
  - *Goal*: Determine which complexity classes are subsets or proper subsets of which other classes (i.e., representing, respectively, "no harder" or "easier" problems)
  - Approach is typically based on *reductions*: proofs that an efficient solution to a problem in one class implies an efficient solution to all problems in another class
- **Typically focus on decision problems**
  - Output is either "yes" or "no"

Venn Diagram of Common Complexity Classes

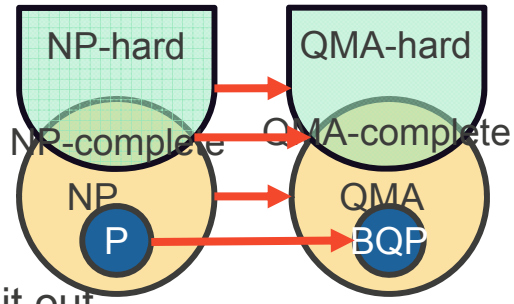


- Problems at least as hard as those in NP
- Not necessarily decision problems
- *Example*: Given a weighted graph, what is the shortest-length Hamiltonian path?
- Hardest of the problems in NP
- *Example*: Given a set of integers, is there a subset whose sum is 0?
- "Hard" decision problems
- Can be solved in polynomial time on a *nondeterministic* Turing machine
- Solutions can be *verified* in polynomial time on a deterministic Turing machine
- *Example*: Does a given integer have a prime factor whose last digit is 3?
- "Easy" decision problems
- Can be solved in polynomial time on a deterministic Turing machine
- *Example*: Does a given matrix have an eigenvalue equal to 1.2?

Quantum Complexity Classes



What Do We Know?



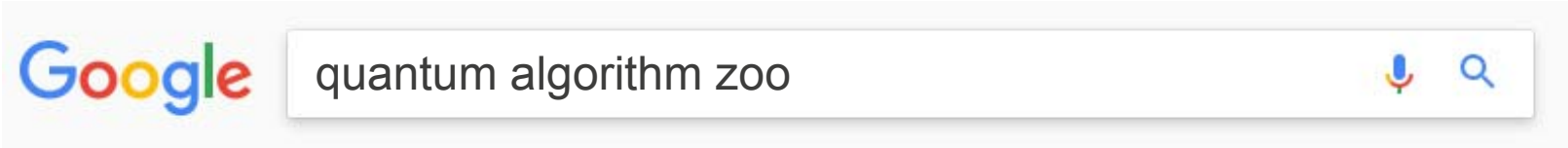
- **Short answer: Almost nothing**
- **P vs. NP**
  - We know that P $\subseteq$ NP, but we don't know whether P = NP or P ≠ NP; conjectured that P ≠ NP
  - $1M prize from the Clay Mathematics Institute if you figure it out
- **NP-intermediate vs. NP-complete**
  - (NP-intermediate are the set of problems in NP but not in NP-complete)
  - We know that NP-intermediate $\subseteq$ NP-complete, but we don't know if they're equal
  - *Implication*: If NP-intermediate ≠ NP-complete, then factoring (NP-intermediate) may in fact be an easy problem, but we just haven't found a good classical algorithm yet
- **P vs. BQP**
  - We know that P $\subseteq$ BQP, but we don't know whether P = BQP or P ≠ BQP
  - *Implication*: If P = BQP, then quantum computers offer no substantial (i.e., superpolynomial) performance advantage over classical computers
- **NP-complete vs. BQP**
  - We don't know relation of BQP to NP-complete; conjectured that BQP $\subset$ NP-complete
  - *Implication*: Believed that quantum computers cannot solve NP-complete problems in polynomial time

It's Not All Doom and Gloom

- **Sure, quantum computers probably can't solve NP-complete problems in polynomial time**
- **Still, even a polynomial-time improvement is better than nothing**
- **Grover's algorithm**
  - Find an item in an unordered list
  - $O(n) \rightarrow O(\sqrt{n})$
- **Shor's algorithm**
  - Factor an integer into primes (NP, but not NP-complete)
  - $O(2^{\sqrt[3]{n}}) \rightarrow O((\log n)^3)$

Aside: Quantum Algorithms (Circuit Model)

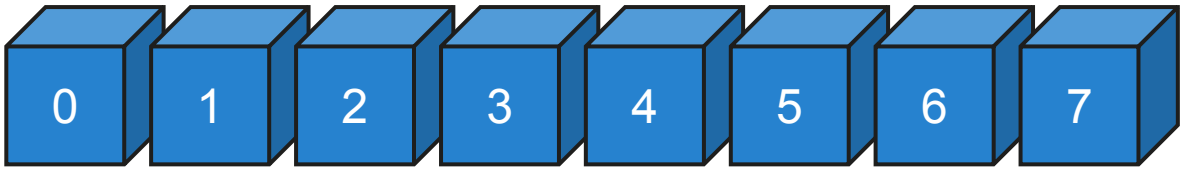
- **Key concepts**
  - *N* classical bits go in, *N* classical bits come out
  - Can operate on all 2^N possibilities in between
  - Requirement: Computation must be reversible (not a big deal in practice)
  - Main challenge: You get only one measurement; how do you know to measure the answer you're looking for?
  - High-level approach: Quantum states based on complex-valued probability *amplitudes*, not probabilities—can sum to 0 to make a possibility go away
- **Very difficult in practice**
  - Only 55 algorithms known to date



  - Based on only a handful of building blocks
  - Each requires substantial cleverness; not much in the way of a standard approach

Grover's Algorithm

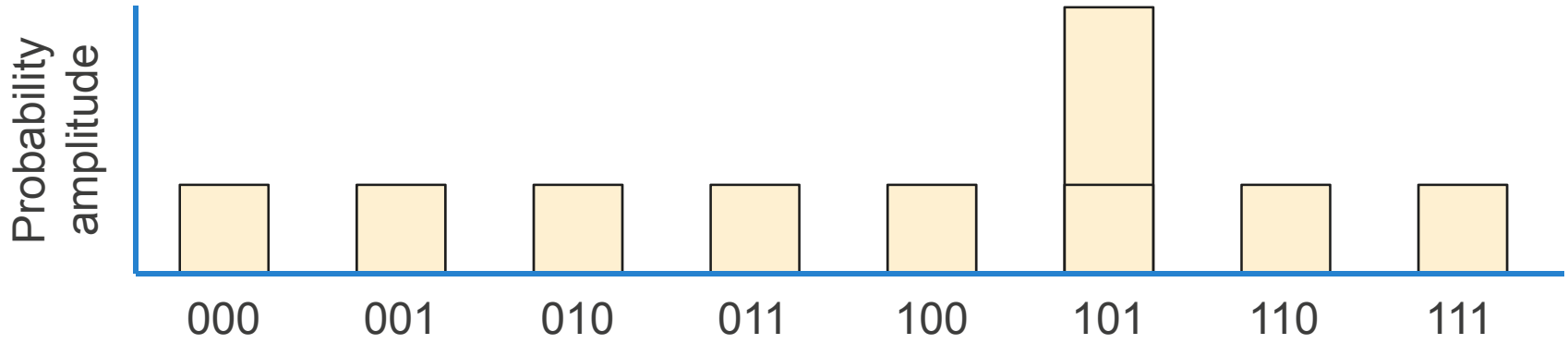
- **Which box contains the prize?**



  – Classically, must open all 8 boxes in the worst case
- **Let's see how we can use quantum effects to do better than that…**
- **Given**
  – A power-of-two number of boxes
  – A guarantee that exactly one box contains the prize
  – An operator U_ω that, given a box number $|x\rangle$, flips the probability amplitude iff the box contains the prize (i.e., $U_\omega|x\rangle = -|x\rangle$ for $x = \omega$ and $U_\omega|x\rangle = |x\rangle$ for $x \neq \omega$)
- **Define the *Grover diffusion operator* as follows**
  – $|s\rangle \equiv \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |x\rangle$ (i.e., the equal superposition of all states)
  – $U_s \equiv 2|s\rangle\langle s| - I$ (the Grover diffusion operator)

Grover's Algorithm (cont.)

- **The basic algorithm is fairly straightforward to apply:**
  - Put each of the *N* qubits in a superposition of $|0\rangle$ and $|1\rangle$
  - For $\sqrt{N}$ iterations
    - Apply U_ω to the state
    - Apply U_s to the state
- **How does that work?**
  - Gradually shifts the probability amplitude to qubit ω from all the other qubits
  - When we measure, we'll get a result of ω with near certainty

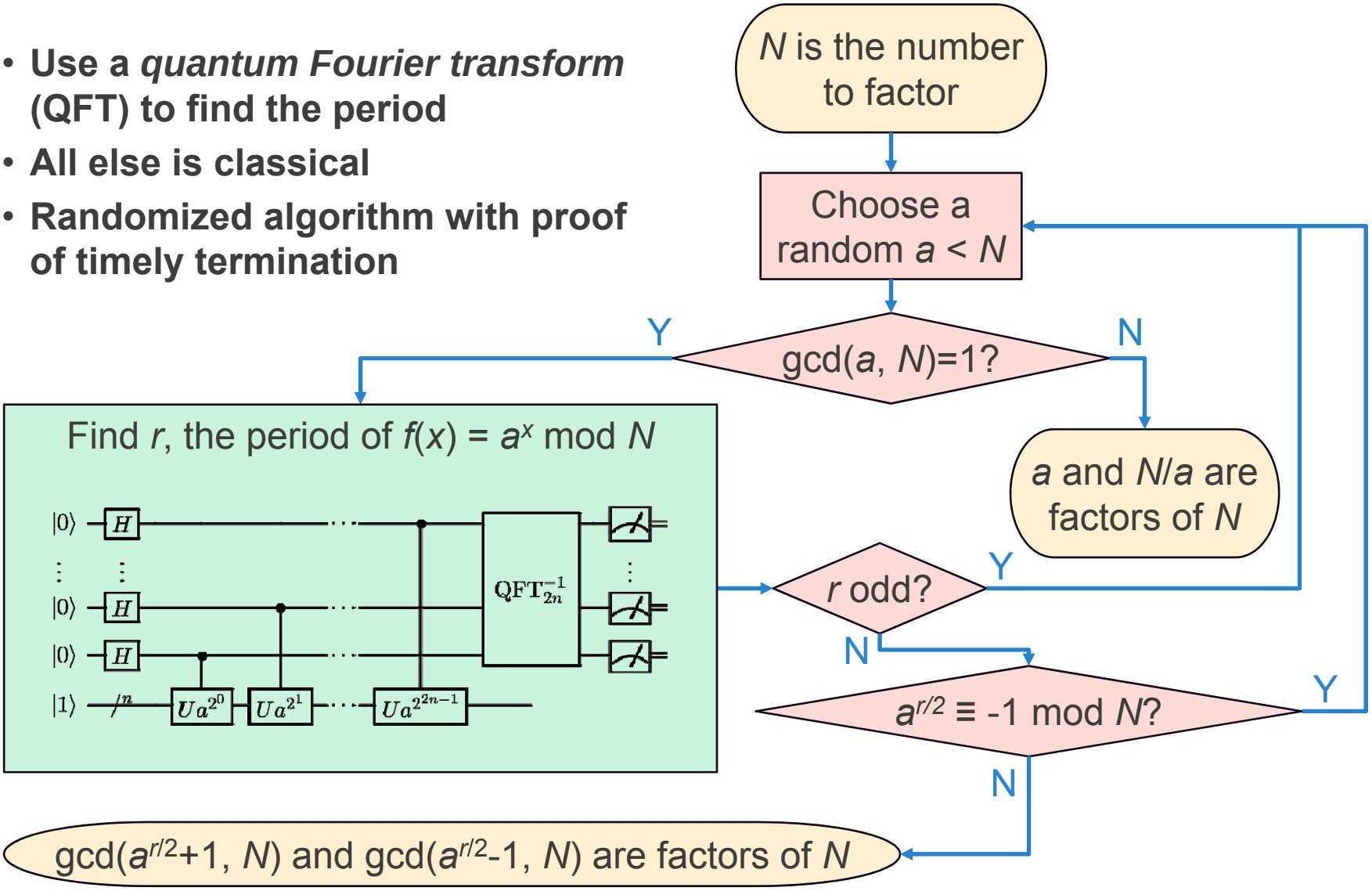


Shor's Algorithm

- **Factor 1,274,093,332,123,426,680,869 into a product of two primes**
  - Okay, it's 135,763,451,261×9,384,656,329
- **Observations**
  - Given that N is the product of two primes, p and q
  - Given some a that is divisible by neither p nor q
  - Then the sequence $\{a^1 \bmod N, a^2 \bmod N, a^3 \bmod N, a^4 \bmod N, a^5 \bmod N, \ldots\}$ will repeat every r elements (the sequence's *period*)
  - As Euler discovered (~1760), r always divides $(p-1)(q-1)$
- **Example**
  - Let a be 2 and N be 15 (=3×5)
  - Then $a^x \bmod N = \{2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1, 2, 4, 8, 1 \ldots\}$ so r is 4
  - Lo and behold, 4 divides (3−1) (5−1)=8
- **Approach**
  - Once we know the period, r, it's not too hard to find N's prime factors p and q
  - Unfortunately, finding r is extremely time-consuming…for a classical computer

Shor's Algorithm (cont.)

- **Use a *quantum Fourier transform* (QFT) to find the period**
- **All else is classical**
- **Randomized algorithm with proof of timely termination**

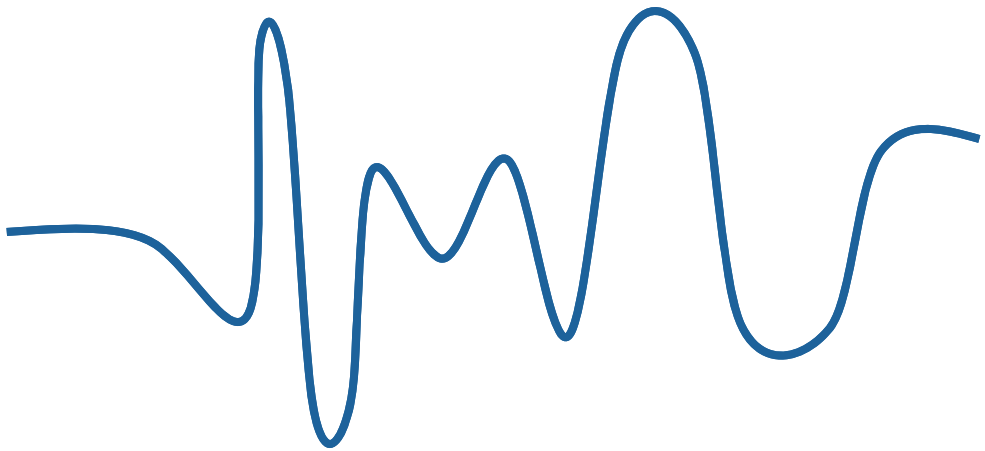


Outline

- Performance potential of quantum computing
- **Quantum annealing**
- Case study: D-Wave quantum annealers
- How to program a quantum annealer
- Parting thoughts

Simulated Annealing

- **Classical (and classic) optimization approach**
- **Find the coordinates of the minimum value in an energy landscape**
- **Conceptual approach**
  - Drop a bunch of rubber balls on the landscape, evaluating the function wherever they hit
  - Hope that one of the balls will bounce and roll downhill to the global minimum
- **Challenge: Commonly get stuck in a local minimum**



Quantum Mechanics to the Rescue

- **Consider adding a time-dependent transverse field to a 2-local Ising Hamiltonian:**

$\mathcal{H}_0$ (classical part)

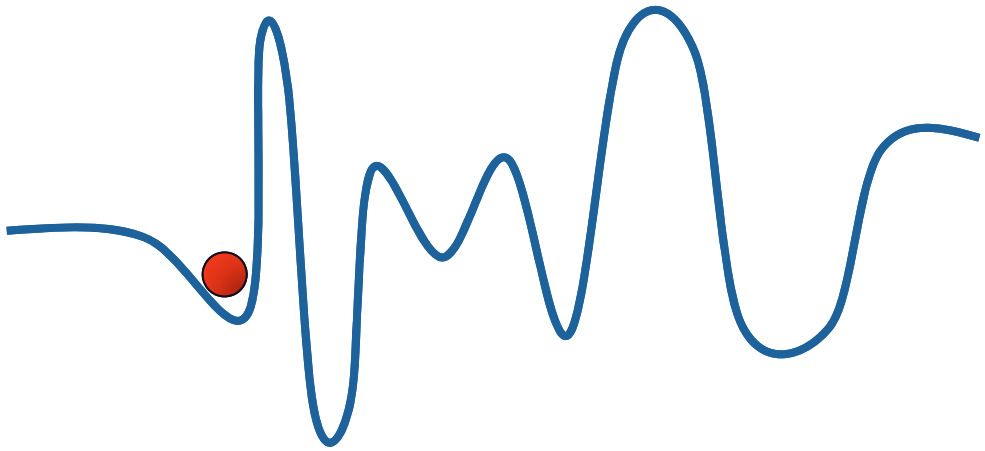
$$\mathcal{H}(t) = -\sum_{i=0}^{N-2}\sum_{j=i+1}^{N-1} J_{i,j}\sigma_i^z\sigma_j^z - \sum_{i=0}^{N-1} h_i\sigma_i^z - \Gamma(t)\sum_{i=0}^{N-1}\sigma_i^x$$

Longitudinal interactions | Longitudinal field | Transverse field

- **Implication of the adiabatic theorem**
  - If we gradually decrease the amplitude of the transverse field, $\Gamma(t)$, from a very large value to 0, we should drive the system into the ground state of $\mathcal{H}_0$
- **The real benefit: quantum tunneling**

Quantum Tunneling

- **Introduced by the $\Gamma(t)$ (transverse) term**
- **Enables jumping from one classical state (eigenstate of $\mathcal{H}_0$) to another**
  - Decreases likelihood of getting stuck in a local minimum
- **Unlike simulated annealing, width of energy barrier is important, but height is not**



Time Evolution

- **If purely adiabatic and sufficiently slow, the system remains in the ground state as it moves from the initial, "generic" Hamiltonian to the problem Hamiltonian**
- **D-Wave's initial state**
  - Ground state (not degenerate): $|+\rangle|+\rangle|+\rangle\cdots|+\rangle$
  - 1st excited state ($\binom{N}{1}$-way degenerate): $|-\rangle|+\rangle|+\rangle\cdots|+\rangle$, $|+\rangle|-\rangle|+\rangle\cdots|+\rangle$, $|+\rangle|+\rangle|-\rangle\cdots|+\rangle$, … $|+\rangle|+\rangle|+\rangle\cdots|-\rangle$
  - 2nd excited state ($\binom{N}{2}$-way degenerate): $|-\rangle|-\rangle|+\rangle\cdots|+\rangle$, $|-\rangle|+\rangle|-\rangle\cdots|+\rangle$, $|+\rangle|-\rangle|-\rangle\cdots|+\rangle$, … $|+\rangle|+\rangle|+\rangle\cdots|-\rangle$
  - etc.

A Brief Aside

- **What we just saw is adiabatic quantum *optimization***
  - Optimization problem is to find the $\sigma_i^z \in \{-1, +1\}$ that minimize $\mathcal{H}_0$
- **A more powerful variation is adiabatic quantum *computing***

$$\mathcal{H}_{ZZXX} = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} J_{i,j} \sigma_i^z \sigma_j^z + \sum_{i=0}^{N-1} h_i \sigma_i^z + \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} K_{i,j} \sigma_i^x \sigma_j^x + \sum_{i=0}^{N-1} \Delta_i \sigma_i^x$$

  - "[A]diabatic quantum computation (error free) is equivalent to the quantum circuit model (error free). So adiabatic quantum computers (error free) are quantum computers (error free) in the most traditional sense."

— Dave Bacon, 27Feb2007

- **In this talk we'll be considering only adiabatic quantum optimization**
  - That's all that's been built to date (at least at large scale)

Annealing Time

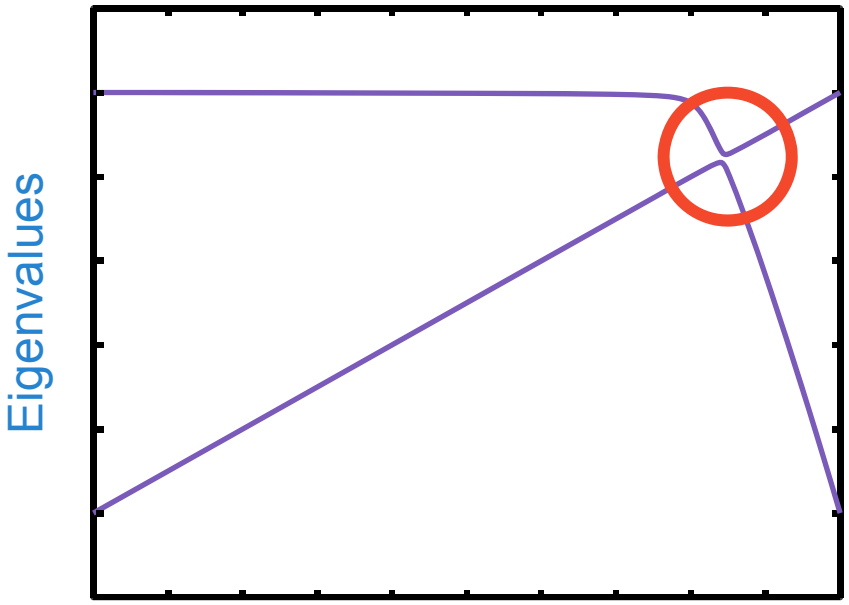
- **From a few slides back:**
  - If we gradually decrease the amplitude of the transverse field, $\Gamma(t)$, from a very large value to 0, we should drive the system into the ground state of $\mathcal{H}_0$
- **What does "gradually" mean?**
  - (Explanation from Farhi and Gutmann)
  - $\mathcal{H}(t)$ encodes our problem
  - Want to evolve the system according to Schrödinger, $i\frac{d}{dt}|\psi\rangle = \mathcal{H}(t)|\psi\rangle$
  - Given that $\mathcal{H}(t)$ has one eigenvalue $E \neq 0$ and the rest 0, find the eigenvector $|w\rangle$ with eigenvector E
  - Assume we're given an orthonormal basis $\{|a\rangle\}$ with $a = 1, \ldots, N$ and that $|w\rangle$ is one of those N basis vectors
  - Let $|s\rangle = \frac{1}{\sqrt{N}}\sum_{a=1}^{N}|a\rangle$
  - We consider the Hamiltonian $\mathcal{H} = E|w\rangle\langle w| + E|s\rangle\langle s|$ (i.e., problem + driver)
  - Let $x = \langle s|w\rangle$
  - Then, omitting a lot of math, we wind up with the probability at time t of finding the state $|w\rangle$ being $\Pr(t) = \sin^2(Ext) + x^2\cos^2(Ext)$
  - To find state $|w\rangle$ with (near) certainty we need to run for time $t_m = \frac{\pi}{2Ex}$

Determining the Annealing Time

- **Unfortunately, we don't in general know how long we need to run (i.e., we can't quickly compute t_m)**
- **Function of the minimum gap between the two smallest eigenvalues at any point during the Hamiltonian's time evolution**
- **Gap can get quite small**
- **Grover's search (right)**
  - Find an *n*-bit number such that

$$\mathcal{H}_P|z\rangle = \begin{cases} |z\rangle & \text{if } z \neq w \\ 0 & \text{if } z = w \end{cases}$$

    for some black-box Hamiltonian $\mathcal{H}_P$
  - Here, $g_{\min} \simeq 2^{1-\frac{n}{2}}$ for n bits
  - Implication: Solution time is $O(2^n)$—no better than classical brute force



Two lowest eigenvalues for a Grover search, 12 bits

Image credit: Farhi, Goldstone, Gutmann, and Sipser (2000)

Annealing Time: Discussion

The bad

- **Very difficult to analyze an algorithm's computational complexity**
  - Need to know the gap between the ground state and first excited state, which can be costly to compute
  - In contrast, circuit-model algorithms tend to be more straightforward to analyze
- **Unknown if quantum annealing can outperform classical**
  - If gap always shrinks exponentially then no
  - (Known that in adiabatic quantum *computing* the gap shrinks polynomially)

Annealing Time: Discussion (cont.)

The good

- **Constants do matter**
  - If the gap is such that a correct answer is expected only once every million anneals, and an anneal takes 5μs, that's still only 5s to get a correct answer—may be good enough
  - On current systems, the gap scaling may be less of a problem than the number of available qubits
- **We may be able to (classically) patch the output to get to the ground state**
  - Hill climbing or other such approaches may help get quickly from a near-ground-state solution into the ground state
- **We may not even need the exact ground state**
  - For many optimization problems, "good and fast" may be preferable to "perfect but slow"

Outline

- Performance potential of quantum computing
- Quantum annealing
- **Case study: D-Wave quantum annealers**
- How to program a quantum annealer
- Parting thoughts

D-Wave's Hamiltonian

- **Problem Hamiltonian (longitudinal field):**

$$\mathcal{H}_P = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} J_{i,j} \sigma_i^z \sigma_j^z + \sum_{i=0}^{N-1} h_i \sigma_i^z$$

Note:
This is a *classical* 2-local Ising Hamiltonian

  - The programmer specifies the $J_{i,j}$ and h_i, and the system solves for the σ_i^z
  - $\sigma_i^z \in \{-1, +1\}$
  - Nominally, $J_{i,j} \in \mathbb{R}$ and $h_i \in \mathbb{R}$, but the hardware limits these to a small set of distinguishable values in the ranges $J_{i,j} \in [-1, +1]$ and $h_i \in [-2, +2]$

- **Application of the time-dependent transverse field:**

$$\mathcal{H}_S(s) = \frac{\varepsilon(s)}{2} \mathcal{H}_P - \frac{\Delta(s)}{2} \sum_{i=0}^{N-1} \sigma_i^x$$

  - Programmer specifies the total annealing time, $T \in [5{,}2000]$ μs
  - $s = t/T$ (i.e., time normalized to [0, 1])
  - $\varepsilon(s)$ and $\Delta(s)$ are scaling parameters (not previously user-controllable but most recent hardware provides a modicum of control over the shape)

D-Wave's Annealing Schedule

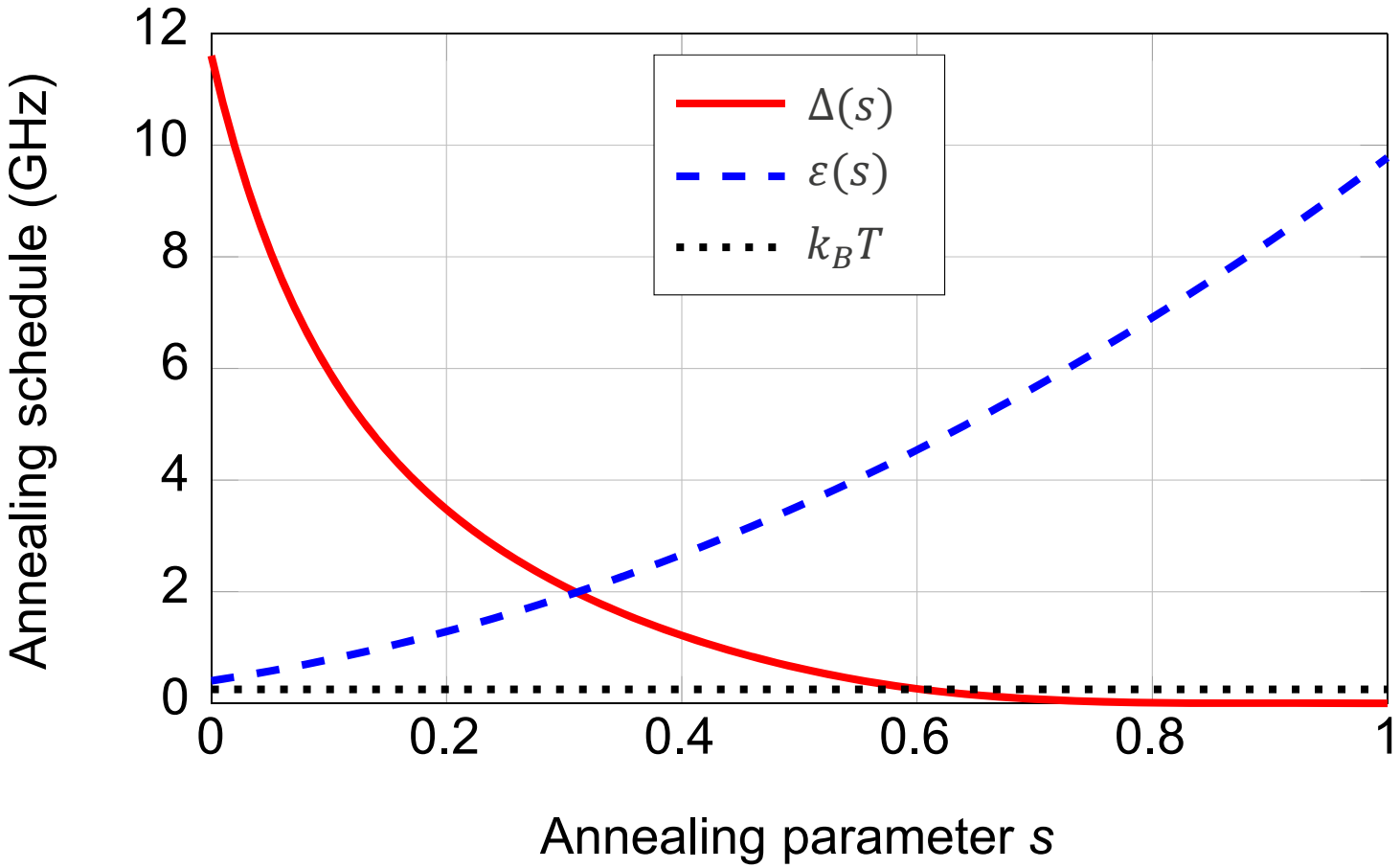
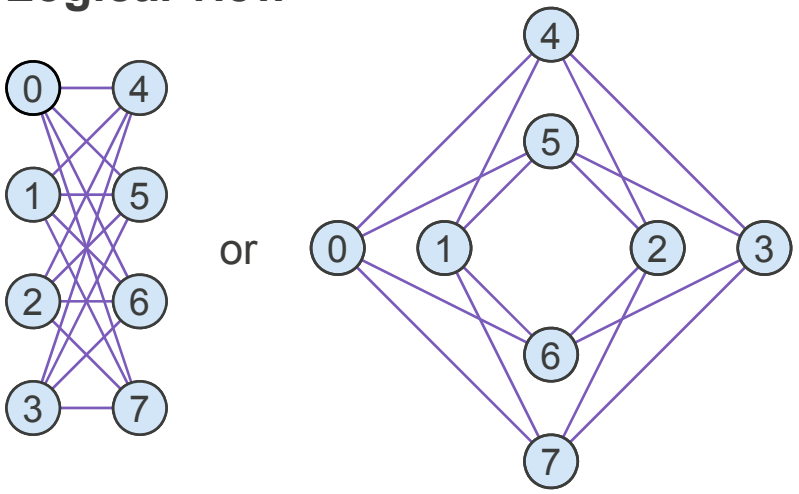


Image credit: King, Hoskinson, Lanting, Andriyash, and Amin (2016)

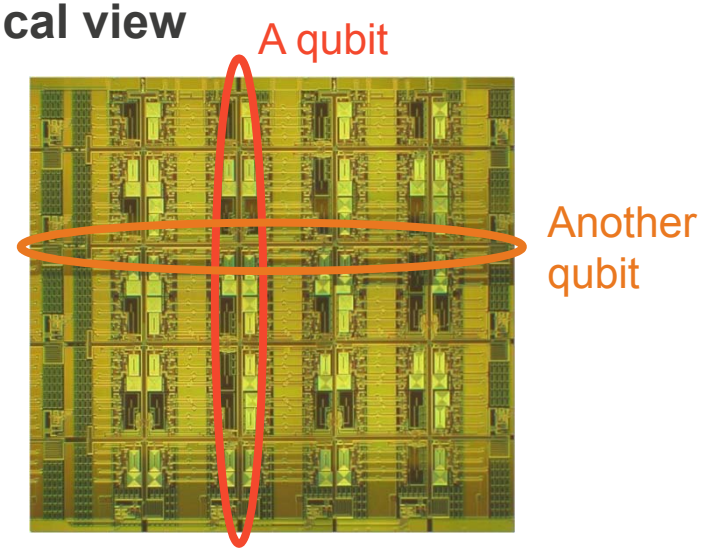
Building Block: The Unit Cell

- **Logical topology**
  - 8 qubits arranged in a bipartite graph
- **Physical implementation**
  - Based on rf-SQUIDs
  - Flux qubits are long loops of superconducting wire interrupted by a set of Josephson junctions (weak links in superconductivity)
  - "Supercurrent" of Cooper pairs of electrons, condensed to a superconducting condensate, flows through the wires
  - Large ensemble of these pairs behaves as a *single* quantum state with net positive or net negative flux
  - …or a superposition of the two (with tunneling)
  - Entanglement introduced at qubit intersections

- **Logical view**

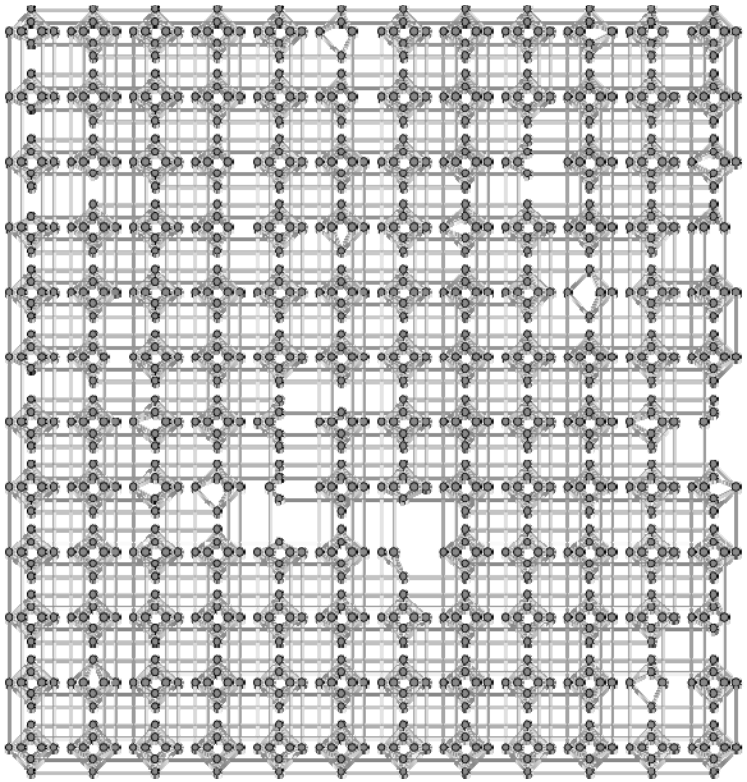


- **Physical view**

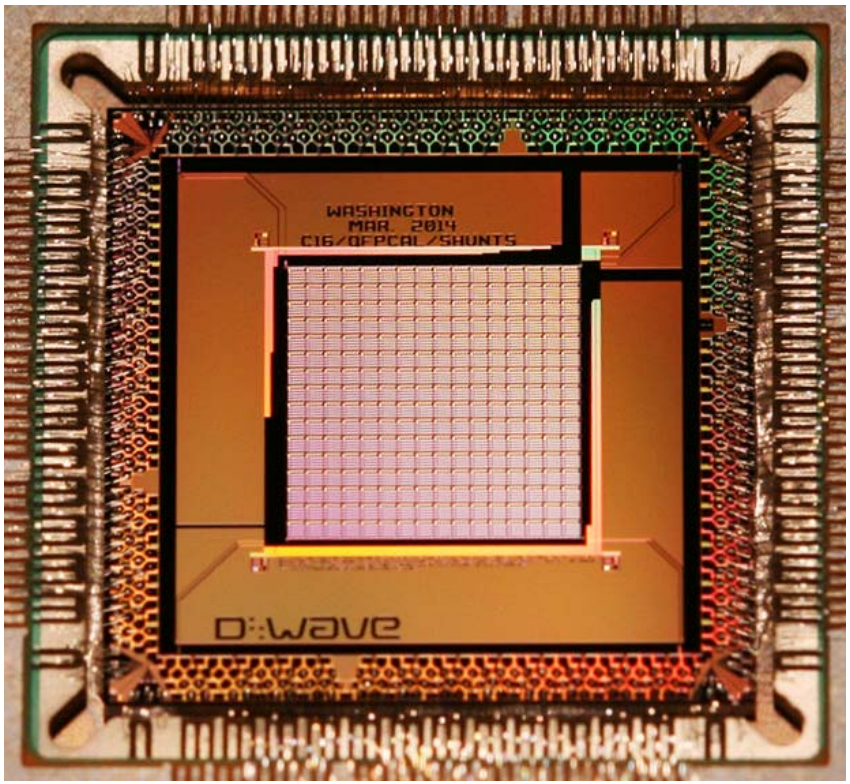


A Complete Chip

- **Logical view**
  - "Chimera graph": 16×16 unit-cell grid
  - Qubits 0–3 couple to north/south neighbors; 4–7 to east/west
  - Inevitably incomplete

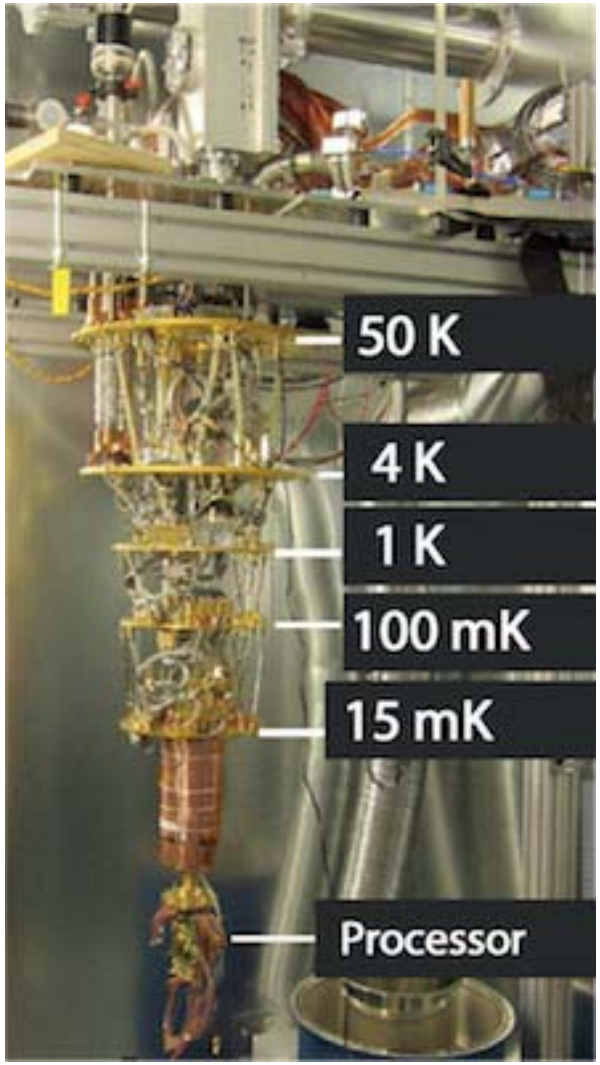


- **Physical view**
  - Chip is about the size of a small fingernail
  - Can even make out unit cells with the naked eye



Cooling

- **Chip must be kept extremely cold for the macroscopic circuit to behave like a two-level (qubit) system**
  - *Much* below the superconducting transition temperature (9000 mK for niobium)
- **Dilution refrigerator**
- **Nominally runs at 15 mK**
- **LANL's D-Wave 2X happens to run at 10.45 mK**
  - That's 0.01°C above absolute zero
  - For comparison, interstellar space is far warmer: 2700 mK



What You Actually See

- **A big, black box**
  - 10’×10’×12’ (3m×3m×3.7m)
  - Mostly empty space
  - Radiation shielding, dilution refrigerator, chip + enclosure, cabling, tubing
  - LANL also had to add a concrete slab underneath to reduce vibration
- **Support logic**
  - Nondescript classical computers
  - Send/receive network requests, communicate with the chip, etc.

Deviation from the Theoretical Model

- **No all-to-all connectivity**
  - Each qubit can be directly coupled to at most 6 other qubits
  - Many qubits and couplers are absent (in an irregular, installation-specific pattern)
- **Not running at absolute zero**
- **Not running in a perfect vacuum**
- **No error correction**
- **We can therefore think of our Hamiltonian as being**

$$\mathcal{H}_S(s) = \frac{\varepsilon(s)}{2}\left(\sum_{\langle i,j\rangle} J_{i,j}\sigma_i^z\sigma_j^z + \sum_{\langle i\rangle} h_i\sigma_i^z\right) - \frac{\Delta(s)}{2}\sum_{\langle i\rangle} h_i\sigma_i^x + \mathcal{H}_?(s)$$

- **in which $\mathcal{H}_?(s)$ encapsulates the interaction with the environment**
  - That is, all the things we don't know and can't practically measure
  - Nonlinear and varies from run to run
- **Also, it takes time to set up a problem and get the results back**
  - *Before*: reset + programming + post-programming thermalization
  - *After*: readout
  - Currently, these dominate the annealing time by many orders of magnitude

Outline

- Performance potential of quantum computing
- Quantum annealing
- Case study: D-Wave quantum annealers
- **How to program a quantum annealer**
- Parting thoughts

The Quantum Optimization Problem

- **We work with only the problem Hamiltonian:**

$$\mathcal{H}_P = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} J_{i,j} \sigma_i^z \sigma_j^z + \sum_{i=0}^{N-1} h_i \sigma_i^z$$

- **Goal (what the hardware does)**
  - Minimize $\sigma_i \in \{-1, +1\}$ subject to provided $J_{i,j} \in \mathbb{R}$ and $h_i \in \mathbb{R}$ coefficients
  - In other words, a quantum optimization program is merely a list of $J_{i,j}$ and h_i
- **Classical**
  - Much easier to reason about (I find) than a quantum Hamiltonian
  - Quantum effects are used internally to work towards the goal
- **2-local**
  - Possible to map >2-local problems into this form at the cost of extra qubits
- **Sparsely connected**
  - Possible to map fully connected problems onto the D-Wave's Chimera graph, again, at the cost of extra qubits

Interpreting the Problem Hamiltonian

- **Let's start by considering only the external field:**

$$\mathcal{H}_P = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} J_{i,j} \sigma_i^z \sigma_j^z + \sum_{i=0}^{N-1} h_i \sigma_i^z$$

- **We arbitrarily call $\sigma_i^z = +1$ "TRUE" and $\sigma_i^z = -1$ "FALSE"**
- **Here are the optimal values of σ_i^z for different values of h_i:**

Negative (say, $h_i = -5$)

σ_i^z	$h_i\sigma_i^z$
–1	+5
+1	–5

Zero

σ_i^z	$h_i\sigma_i^z$
–1	0
+1	0

Positive (say, $h_i = +5$)

σ_i^z	$h_i\sigma_i^z$
–1	–5
+1	+5

- **Observations**
  - A negative h_i means, "I want σ_i^z to be TRUE"
  - A zero h_i means, "I don't care if σ_i^z is TRUE or FALSE"
  - A positive h_i means, "I want σ_i^z to be FALSE"

Interpreting the Problem Hamiltonian (cont.)

- **Now let's consider only the coupler strengths:**

$$\mathcal{H}_P = \sum_{i=0}^{N-2} \sum_{j=i+1}^{N-1} J_{i,j} \sigma_i^z \sigma_j^z + \sum_{i=0}^{N-1} h_i \sigma_i^z$$

- **Here are the optimal values of σ_i^z and σ_j^z for different values of $J_{i,j}$:**

Negative ($J_{i,j} = -5$)

σ_i^z	σ_j^z	$J_{i,j}\sigma_i^z\sigma_j^z$
–1	–1	–5
–1	+1	+5
+1	–1	+5
+1	+1	–5

Zero

σ_i^z	σ_j^z	$J_{i,j}\sigma_i^z\sigma_j^z$
–1	–1	0
–1	+1	0
+1	–1	0
+1	+1	0

Positive ($J_{i,j} = +5$)

σ_i^z	σ_j^z	$J_{i,j}\sigma_i^z\sigma_j^z$
–1	–1	+5
–1	+1	–5
+1	–1	–5
+1	+1	+5

- **Observations**
  - A negative $J_{i,j}$ means, "I want σ_i^z and σ_j^z to be equal"
  - A zero $J_{i,j}$ means, "I don't care how σ_i^z and σ_j^z are related"
  - A positive $J_{i,j}$ means, "I want σ_i^z and σ_j^z to be different"

Solving a Map-Coloring Problem

- **Given a planar map, color each region with one of four colors such that no two adjacent regions have the same color**
  - NP-hard problem
- **We start by defining a region as having exactly one color**
  - Let's use a unary encoding with +1 ≡ has the color and –1 ≡ lacks the color

σ_{red}	σ_{yellow}	σ_{green}	σ_{blue}
–1	–1	–1	–1
–1	–1	–1	+1
–1	–1	+1	–1
–1	–1	+1	+1
–1	+1	–1	–1
–1	+1	–1	+1
–1	+1	+1	–1
–1	+1	+1	+1

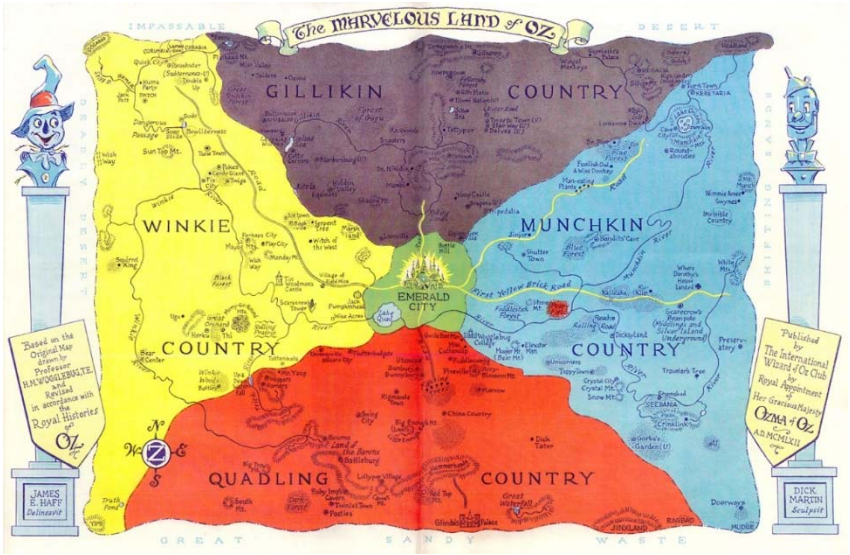
σ_{red}	σ_{yellow}	σ_{green}	σ_{blue}
+1	–1	–1	–1
+1	–1	–1	+1
+1	–1	+1	–1
+1	–1	+1	+1
+1	+1	–1	–1
+1	+1	–1	+1
+1	+1	+1	–1
+1	+1	+1	+1

A Hamiltonian for a Region of a Map

- **Define a system of inequalities**
- **Ground state (four-way degenerate)**
  - $\mathcal{H}(---+) = \mathcal{H}(--+-) = \mathcal{H}(-+--) = \mathcal{H}(+---) = k$
- **All excited states**
  - $\mathcal{H}(----) > k$
  - $\mathcal{H}(--++) > k$
  - $\mathcal{H}(-+-+) > k$
  - $\mathcal{H}(-++-) > k$
  - $\mathcal{H}(-+++) > k$
  - $\mathcal{H}(+--+) > k$
  - $\mathcal{H}(+-+-) > k$
  - $\mathcal{H}(+-++) > k$
  - $\mathcal{H}(++--) > k$
  - $\mathcal{H}(++-+) > k$
  - $\mathcal{H}(+++-) > k$
  - $\mathcal{H}(++++) > k$
- **Solve for the h_i and $J_{i,j}$ coefficients**
  - $\mathcal{H}(\sigma_r, \sigma_y, \sigma_g, \sigma_b) = h_r\sigma_r + h_y\sigma_y + h_g\sigma_g + h_b\sigma_b + J_{r,y}\sigma_r\sigma_y + J_{r,g}\sigma_r\sigma_g + J_{r,b}\sigma_r\sigma_b + J_{y,g}\sigma_y\sigma_g + J_{y,b}\sigma_y\sigma_b + J_{g,b}\sigma_g\sigma_b$
- **One possible solution (not unique)**
  - $\mathcal{H}(\sigma_r, \sigma_y, \sigma_g, \sigma_b) = \sigma_r + \sigma_y + \sigma_g + \sigma_b + \frac{1}{2}\sigma_r\sigma_y + \frac{1}{2}\sigma_r\sigma_g + \frac{1}{2}\sigma_r\sigma_b + \frac{1}{2}\sigma_y\sigma_g + \frac{1}{2}\sigma_y\sigma_b + \frac{1}{2}\sigma_g\sigma_b$

A Hamiltonian for the Complete Map-Coloring Problem

- **Hamiltonians are additive**
  - We can add up a bunch of region Hamiltonians to produce a map Hamiltonian
- **Use antiferromagnetic couplings ($J_{i,j} > 0$) to avoid assigning adjacent regions the same color***
  - $\sigma_{r_{WC}}\sigma_{r_{GC}} + \sigma_{y_{WC}}\sigma_{y_{GC}} + \sigma_{g_{WC}}\sigma_{g_{GC}} + \sigma_{b_{WC}}\sigma_{b_{GC}} + \sigma_{r_{EC}}\sigma_{r_{MC}} + \sigma_{g_{EC}}\sigma_{g_{MC}} + \cdots$



$\mathcal{H}(\sigma_{r_{GC}}, \sigma_{y_{GC}}, \sigma_{g_{GC}}, \sigma_{b_{GC}})$

$\mathcal{H}(\sigma_{r_{WC}}, \sigma_{y_{WC}}, \sigma_{g_{WC}}, \sigma_{b_{WC}})$

$\mathcal{H}(\sigma_{r_{EC}}, \sigma_{y_{EC}}, \sigma_{g_{EC}}, \sigma_{b_{EC}})$

$\mathcal{H}(\sigma_{r_{MC}}, \sigma_{y_{MC}}, \sigma_{g_{MC}}, \sigma_{b_{MC}})$

$\mathcal{H}(\sigma_{r_{QC}}, \sigma_{y_{QC}}, \sigma_{g_{QC}}, \sigma_{b_{QC}})$

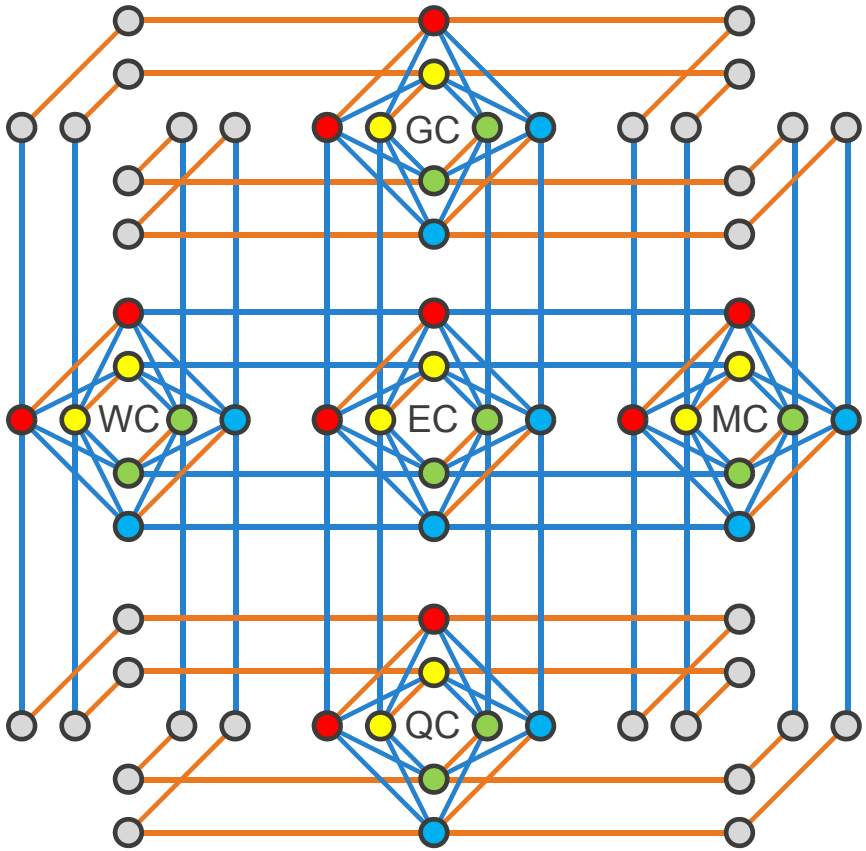
*Oversimplification: OK if neither of two adjacent regions has a given color. Adding $\mathcal{H} = \sigma_i + \sigma_j + \sigma_i\sigma_j$ should do the trick.

Embedding the Problem in a Chimera Graph

- **Each qubit in a region needs to couple with all three other qubits**

  and

- **EC needs to be able to couple to the north (GC), south (QC), east (MC), and west (WC)**
  - Solution: Split each qubit into two ferromagnetically coupled ($J_{i,j} < 0$) qubits
  - One qubit couples north/south and one qubit couples east/west
- **All regions except EC need to be able to couple diagonally**
  - Solution: Introduce "ghost" unit cells solely for routing
  - Alternative: Replicate regions (two unit cells for each region but EC) and couple ferromagnetically



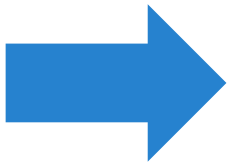
Is There an Easier Approach?

- **Yes!**
- **My personal research interest: How to compile classical computer programs into a 2-local Ising Hamiltonian**
- **Be patient; it's going to take a bunch of slides to get there…**

Step 1: Symbolic Hamiltonians

- **Quantum Macro Assembler (QMASM)**
- **Accept hardware-oblivious Ising Hamiltonians**
  - Do not need to be 2-local
  - Do not need to be mapped to a Chimera graph
  - Do not need to scale the h_i or $J_{i,j}$ into hardware-specific ranges
- **Programmer-friendly**
  - Specify qubits by name, not by number
  - Run on a D-Wave system and report results by name
  - Define sub-Hamiltonians as macros then instantiate those repeatedly
  - Enable "pinning" qubits to TRUE or FALSE (by adding a helper qubit with $h_i \leqslant 0$ and ferromagnetically coupling to that)

$$\mathcal{H}_{\text{region}}(\sigma_r, \sigma_y, \sigma_g, \sigma_b) = \sigma_r + \sigma_y + \sigma_g + \sigma_b + \frac{1}{2}\sigma_r\sigma_y + \frac{1}{2}\sigma_r\sigma_g + \frac{1}{2}\sigma_r\sigma_b + \frac{1}{2}\sigma_y\sigma_g + \frac{1}{2}\sigma_y\sigma_b + \frac{1}{2}\sigma_g\sigma_b$$



```
# Define a region.
!begin_macro region
  red    1.0
  yellow 1.0
  green  1.0
  blue   1.0

  red    yellow  0.5
  red    green   0.5
  red    blue    0.5
  yellow green   0.5
  yellow blue    0.5
  green  blue    0.5
!end_macro region
```

QMASM Code for Map-Coloring the Land of Oz

```
!include <region>

!use_macro region GC
!use_macro region WC
!use_macro region QC
!use_macro region MC
!use_macro region EC

!alias NEQ 1.0

# Adjacent regions must
# use different colors.
GC.red MC.red NEQ
MC.red QC.red NEQ
QC.red WC.red NEQ
WC.red GC.red NEQ
GC.red EC.red NEQ
MC.red EC.red NEQ
QC.red EC.red NEQ
WC.red EC.red NEQ

GC.yellow MC.yellow NEQ
MC.yellow QC.yellow NEQ
QC.yellow WC.yellow NEQ
WC.yellow GC.yellow NEQ
GC.yellow EC.yellow NEQ
MC.yellow EC.yellow NEQ
QC.yellow EC.yellow NEQ
WC.yellow EC.yellow NEQ

GC.green MC.green NEQ
MC.green QC.green NEQ
QC.green WC.green NEQ
WC.green GC.green NEQ
GC.green EC.green NEQ
MC.green EC.green NEQ
QC.green EC.green NEQ
WC.green EC.green NEQ

GC.blue MC.blue NEQ
MC.blue QC.blue NEQ
QC.blue WC.blue NEQ
WC.blue GC.blue NEQ
GC.blue EC.blue NEQ
MC.blue EC.blue NEQ
QC.blue EC.blue NEQ
WC.blue EC.blue NEQ

# Adjust for number of
# neighbors.
!alias outer_neigh 3.0
!alias inner_neigh 4.0

GC.red    outer_neigh
GC.yellow outer_neigh
GC.green  outer_neigh
GC.blue   outer_neigh

MC.red    outer_neigh
MC.yellow outer_neigh
MC.green  outer_neigh
MC.blue   outer_neigh

QC.red    outer_neigh
QC.yellow outer_neigh
QC.green  outer_neigh
QC.blue   outer_neigh

WC.red    outer_neigh
WC.yellow outer_neigh
WC.green  outer_neigh
WC.blue   outer_neigh

EC.red    inner_neigh
EC.yellow inner_neigh
EC.green  inner_neigh
EC.blue   inner_neigh
```

Step 2: Universal Building Blocks

- **Set up and solve systems of inequalities for various Boolean functions**
  - We already have NOT, which is simply $\mathcal{H}_{\text{NOT}}(\sigma_A, \sigma_Y) = \sigma_A \sigma_Y$
  - We also have wires, which are simply $\mathcal{H}_{\text{wire}}(\sigma_A, \sigma_Y) = -\sigma_A \sigma_Y$

AND
$(Y = A \wedge B)$

σ_A	σ_B	σ_Y
–1	–1	–1
–1	–1	+1
–1	+1	–1
–1	+1	+1
+1	–1	–1
+1	–1	+1
+1	+1	–1
+1	+1	+1

OR
$(Y = A \vee B)$

σ_A	σ_B	σ_Y
–1	–1	–1
–1	–1	+1
–1	+1	–1
–1	+1	+1
+1	–1	–1
+1	–1	+1
+1	+1	–1
+1	+1	+1

XOR
$(Y = A \oplus B)$

σ_A	σ_B	σ_Y
–1	–1	–1
–1	–1	+1
–1	+1	–1
–1	+1	+1
+1	–1	–1
+1	–1	+1
+1	+1	–1
+1	+1	+1

Uh-oh: XOR Has No Solution

- **Introduce an ancilla qubit to make it work**
  - Open problem #1: How many ancilla qubits are needed for an arbitrary truth table?
  - Open problem #2: How should the extra column(s) be populated?
- **Here's an example of an ancilla column that works:**

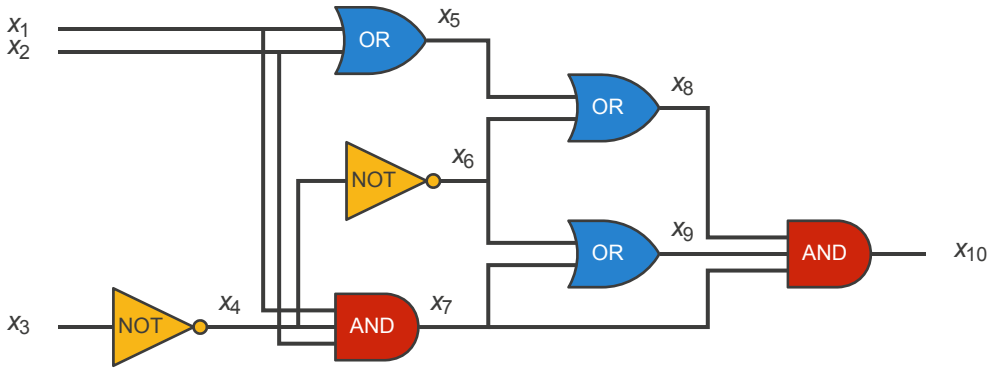
σ_A	σ_B	σ_Y	σ_a
–1	–1	–1	–1
–1	–1	–1	+1
–1	–1	+1	–1
–1	–1	+1	+1
–1	+1	–1	–1
–1	+1	–1	+1
–1	+1	+1	–1
–1	+1	+1	+1

σ_A	σ_B	σ_Y	σ_a
+1	–1	–1	–1
+1	–1	–1	+1
+1	–1	+1	–1
+1	–1	+1	+1
+1	+1	–1	–1
+1	+1	–1	+1
+1	+1	+1	–1
+1	+1	+1	+1

Sample Solutions

- **AND:** $\mathcal{H}_{\wedge}(\sigma_A, \sigma_B, \sigma_Y) = -\frac{1}{2}\sigma_A - \frac{1}{2}\sigma_B + \sigma_Y + \frac{1}{2}\sigma_A\sigma_B - \sigma_A\sigma_Y - \sigma_B\sigma_Y$
- **OR:** $\mathcal{H}_{\vee}(\sigma_A, \sigma_B, \sigma_Y) = \frac{1}{2}\sigma_A + \frac{1}{2}\sigma_B - \sigma_Y + \frac{1}{2}\sigma_A\sigma_B - \sigma_A\sigma_Y - \sigma_B\sigma_Y$
- **XOR:** $\mathcal{H}_{\oplus}(\sigma_A, \sigma_B, \sigma_Y, \sigma_a) = \frac{1}{2}\sigma_A + \frac{1}{2}\sigma_B + \frac{1}{2}\sigma_Y + \sigma_a + \frac{1}{2}\sigma_A\sigma_B + \frac{1}{2}\sigma_A\sigma_Y + \sigma_A\sigma_a + \frac{1}{2}\sigma_B\sigma_Y + \sigma_B\sigma_a + \sigma_Y\sigma_a$

- **Remember: Solutions are not unique; your answers may vary**
- **What we have so far is sufficient to solve satisfiability problems**
  - Given a Boolean function, is there a set of inputs for which the output is TRUE?
  - NP-complete problem



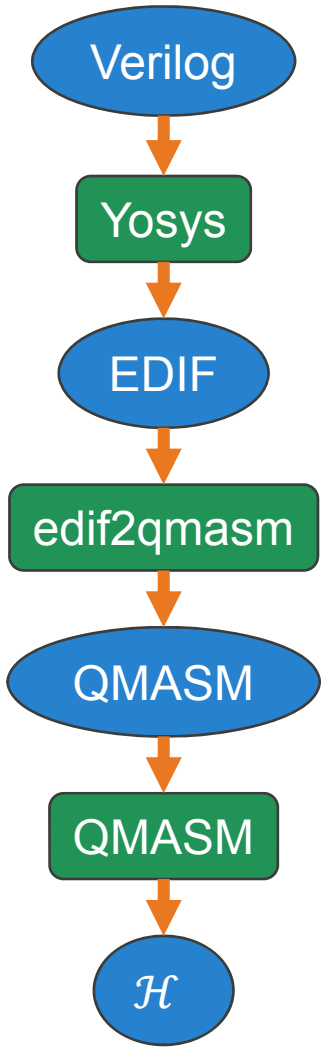
Note: Uses 3-input AND (left as exercise for the reader)

Leveraging Decades of Computer Engineering

- **Today, virtually all non-trivial hardware is created using a *hardware description language* (HDL)**
  - Looks more-or-less like an ordinary programming language
  - Variables, arithmetic operators, relational operators, conditionals, loops, modules, …
- **Hardware synthesis tools compile HDLs to a set of logic primitives**
  - AND, OR, NOT, XOR, …
- **Often perform a variety of transformations to reduce the amount of logic required**

Compiling Hardware to an Ising Hamiltonian

- **Start with a program written in a hardware-description language**
  - E.g., Verilog (1984)
- **Let an existing hardware-synthesis tool compile the HDL to a circuit of Boolean operators**
- **Convert the circuit to QMASM**
- **Generate a D-Wave-specific Ising Hamiltonian**
- **Run on a D-Wave**



Hardware-description language

Hardware-synthesis tool

Netlist (machine-readable circuit description)

Format-conversion tool

Logical Hamiltonian

Quantum macro assembler

Physical Hamiltonian

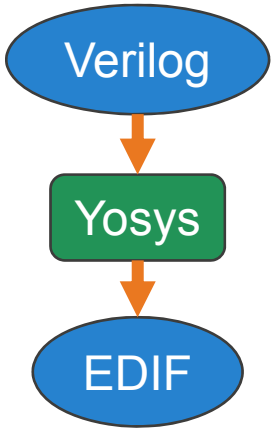
Map Coloring Written in Verilog

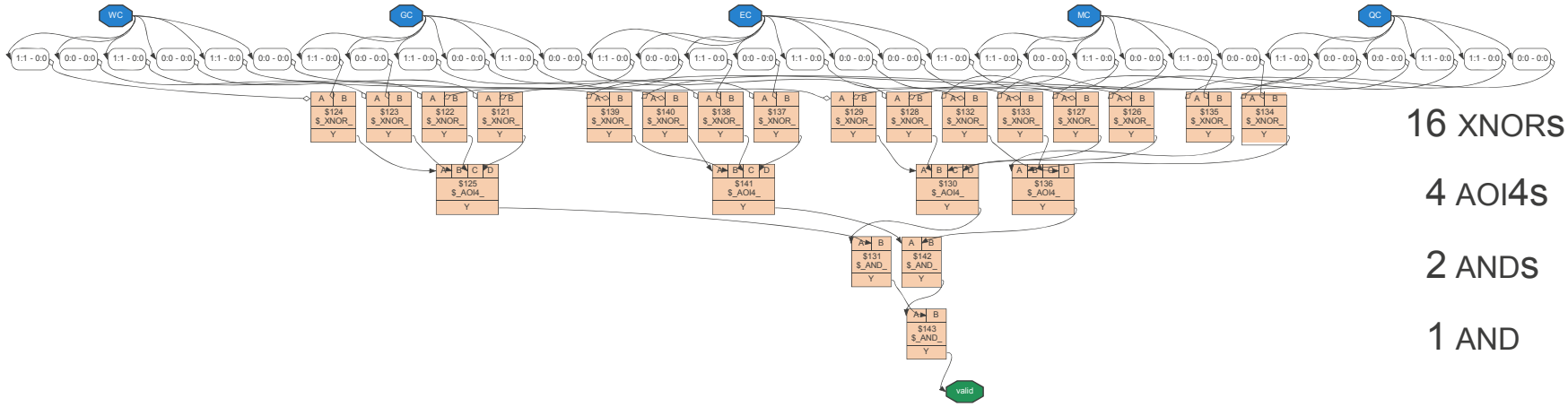
```
module map_color (GC, WC, QC, MC, EC, valid);
  input [1:0] GC;
  input [1:0] WC;
  input [1:0] QC;
  input [1:0] MC;
  input [1:0] EC;
  output valid;
  wire [7:0] tests;

  assign tests[0] = GC != WC;
  assign tests[1] = WC != QC;
  assign tests[2] = QC != MC;
  assign tests[3] = MC != GC;
  assign tests[4] = EC != GC;
  assign tests[5] = EC != WC;
  assign tests[6] = EC != QC;
  assign tests[7] = EC != MC;

  assign valid = &tests[7:0];
endmodule
```

- **Things to note**
  - *Much* shorter than the QMASM code
  - Terminology: a "wire" is an internal variable
  - Each variable has its precise bit-width specified (important when every qubit counts)
  - Can compare multi-bit numbers with ease
- **Key concept**
  - Program is written as a map-coloring validator
  - Given a map coloring, return TRUE if the coloring is valid, FALSE otherwise
  - We're going to run this *backward* by pinning `valid` to TRUE
  - (Can't do that with ordinary hardware)

Map Coloring after Hardware Synthesis

- Yes, this is an eye chart; sorry about that
- **Region colors** (top) progress through a tree of 23 **gates** to produce a **valid bit**





16 XNORS

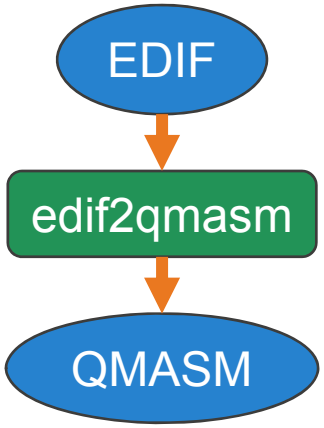
4 AOI4s

2 ANDS

1 AND

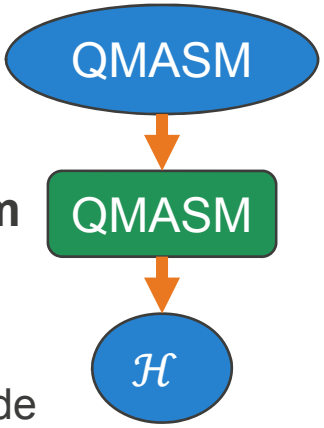
Map Coloring after Conversion to QMASM

```
!include <stdcell>

!begin_macro map_color
  !use_macro AND $id00014
  !use_macro AND $id00025
  !use_macro AND $id00026
  !use_macro AOI4 $id00008
  !use_macro AOI4 $id00013
  !use_macro AOI4 $id00019
  !use_macro AOI4 $id00024
  !use_macro XNOR $id00004
  !use_macro XNOR $id00005
  !use_macro XNOR $id00006
  !use_macro XNOR $id00007
  !use_macro XNOR $id00009
  !use_macro XNOR $id00010
  !use_macro XNOR $id00011
  !use_macro XNOR $id00012
  !use_macro XNOR $id00015
  !use_macro XNOR $id00016
  !use_macro XNOR $id00017
  !use_macro XNOR $id00018
  !use_macro XNOR $id00020
  !use_macro XNOR $id00021
  !use_macro XNOR $id00022
  !use_macro XNOR $id00023
  EC[0] <-> $id00004.B
  EC[1] <-> $id00005.B
  GC[0] <-> $id00011.A
  GC[1] <-> $id00012.A
  MC[0] <-> $id00016.A
  MC[1] <-> $id00015.A
  QC[0] <-> $id00010.A
  QC[1] <-> $id00009.A
  WC[0] <-> $id00004.A
  WC[1] <-> $id00005.A
  $id00004.A = $id00006.A
  $id00004.A = $id00023.B
  $id00004.B = $id00010.B
  $id00004.B = $id00016.B
  $id00004.B = $id00020.B
  $id00005.A = $id00007.A
  $id00005.A = $id00022.B
  $id00005.B = $id00009.B
  $id00005.B = $id00015.B
  $id00005.B = $id00021.B
  $id00006.A = $id00023.B
  $id00007.A = $id00022.B
  $id00008.A = $id00007.Y
  $id00008.B = $id00006.Y
  $id00008.C = $id00005.Y
  $id00008.D = $id00004.Y
  $id00009.A = $id00018.A
  $id00009.A = $id00022.A
  $id00009.B = $id00015.B
  $id00009.B = $id00021.B
  $id00010.A = $id00017.A
  $id00010.A = $id00023.A
  $id00010.B = $id00016.B
  $id00010.B = $id00020.B
  $id00011.A = $id00006.B
  $id00011.A = $id00020.A
  $id00011.B = $id00017.B
  $id00012.A = $id00007.B
  $id00012.A = $id00021.A
  $id00012.B = $id00018.B
  $id00013.A = $id00012.Y
  $id00013.B = $id00011.Y
  $id00013.C = $id00010.Y
  $id00013.D = $id00009.Y
  $id00014.A = $id00013.Y
  $id00014.B = $id00008.Y
  $id00015.A = $id00012.B
  $id00015.A = $id00018.B
  $id00015.B = $id00021.B
  $id00016.A = $id00011.B
  $id00016.A = $id00017.B
  $id00016.B = $id00020.B
  $id00017.A = $id00023.A
  $id00018.A = $id00022.A
  $id00019.A = $id00018.Y
  $id00019.B = $id00017.Y
  $id00019.C = $id00016.Y
  $id00019.D = $id00015.Y
  $id00020.A = $id00006.B
  $id00021.A = $id00007.B
  $id00024.A = $id00023.Y
  $id00024.B = $id00022.Y
  $id00024.C = $id00021.Y
  $id00024.D = $id00020.Y
  $id00025.A = $id00024.Y
  $id00025.B = $id00019.Y
  $id00026.A = $id00025.Y
  $id00026.B = $id00014.Y
  $id00026.Y = valid
  EC[0] = $id00010.B
  EC[0] = $id00016.B
  EC[0] = $id00020.B
  EC[1] = $id00009.B
  EC[1] = $id00015.B
  EC[1] = $id00021.B
  GC[0] = $id00006.B
  GC[0] = $id00020.A
  GC[1] = $id00007.B
  GC[1] = $id00021.A
  MC[0] = $id00011.B
  MC[0] = $id00017.B
  MC[1] = $id00012.B
  MC[1] = $id00018.B
  QC[0] = $id00017.A
  QC[0] = $id00023.A
  QC[1] = $id00018.A
  QC[1] = $id00022.A
  WC[0] = $id00006.A
  WC[0] = $id00023.B
  WC[1] = $id00007.A
  WC[1] = $id00022.B
!end_macro map_color

!use_macro map_color map_color
```



- **Another eye chart; sorry again**
- **Basically, the generated QMASM code is a direct translation of the circuit in the previous slide**
  - Gates → macros
  - Wires → ferromagnetic couplings

Map Coloring as a Physical Hamiltonian



- **One final eye chart…**
- **A Hamiltonian suitable for direct execution on a D-Wave system**
- **Not something a human could easily produce**
  - But that's what computers are for
  - And this all came from ~20 lines of easy-to-write, easy-to-read Verilog code

Outline

- Performance potential of quantum computing
- Quantum annealing
- Case study: D-Wave quantum annealers
- How to program a quantum annealer
- **Parting thoughts**

Quantum Computing Performance

- **Is any quantum computer today faster than a modern classical computer?**
  - No, not for any real problem
  - Always read the fine print (e.g., Google's reported 10^8 speedup on a D-Wave 2X was for a D-Wave-friendly problem running against a non-optimal classical algorithm on a single core)
  - Way too few bits even to *express* sufficiently challenging/interesting problems (e.g., LANL's D-Wave 2X has 1095 bits ≈ 136 bytes)
  - And I didn't even talk about error correction, which can be extremely costly in terms of qubit count
- **Will quantum computers eventually outperform classical computers?**
  - Likely, but based on the little we actually know how to prove, not guaranteed
  - For adiabatic quantum optimization, the answer is particularly murky

"I wouldn't bet against quantum computing."

— Rupak Biswas (NASA)

  - Even if we can't often reduce execution time from $O(2^n)$ to $O(n^k)$, *any* speedup is good speedup

Quantum Computer Programmability

- **No one said this was going to be easy**
- **Circuit-model quantum computing**
  - Programmer directly manipulates quantum effects (as unitary matrix transformations)
  - Art lies in canceling out non-solutions so solutions appear with high probability
  - "High pain, large gain"
- **Adiabatic quantum optimization**
  - Quantum effects not exposed to the programmer (classical Ising Hamiltonian)
  - As we've seen, it's possible to compile classical code into a target Hamiltonian
  - "Low pain, questionable gain"
- **Here's a different way to think about programmability**
  - Suppose, worst case, that adiabatic quantum optimization will *never* reach an exact answer faster than a classical computer
  - Now consider an NP-complete or NP-hard problem
  - Classical brute-force solution: easy to write, slow to run, exact answer
  - Classical heuristic solution: difficult to write, fast to run, approximate answer
  - Quantum-annealing solution: easy to write, fast to run, approximate answer
  - Maybe the key benefit of quantum annealing is "performance per unit effort"

References

- **Complexity theory**
  - Scott Aaronson. “An Invitation to Quantum Complexity Theory: The Study of What We Can’t Do with Computers We Don’t Have”. QIP 2008, New Delhi. URL: http://www.scottaaronson.com/talks/tutorial.ppt.
  - Scott Aaronson. “PHYS771: Quantum Computing Since Democritus (Lecture Notes)”. University of Waterloo, Fall 2006. URL: http://scottaaronson.com/democritus/. See in particular Lecture 6: “P, NP, and Friends”.
- **Quantum annealing**
  - Edward Farhi and Sam Gutmann. “An Analog Analogue of a Digital Quantum Computation”. Physical Review A, **57**(4), 2403. 1 April 1998. DOI: 10.1103/PhysRevA.57.2403.
  - Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Michael Sipser. “Quantum Computation by Adiabatic Evolution”. Technical Report MIT-CTP-2936, 28 January 2000. arXiv:quant-ph/0001106.
  - Dave Bacon. “Orion Into The Future”. The Quantum Pontiff, 2 February 2007. URL: http://dabacon.org/pontiff/?p=1427.
  - Tadashi Kadowaki and Hidetoshi Nishimori. “Quantum Annealing in the Transverse Ising Model”. Physical Review E, **58**(5), 5355. 1 November 1998. DOI: 10.1103/PhysRevE.58.5355.

References (cont.)

- **D-Wave hardware**
  - D-Wave publications and white papers: https://www.dwavesys.com/resources/publications
  - Andrew D. King, Emile Hoskinson, Trevor Lanting, Evgeny Andriyash, and Mohammad H. Amin. "Degeneracy, Degree, and Heavy Tails in Quantum Annealing". Physical Review A, **93**(5), 052320, 18 May 2016. DOI: 10.1103/PhysRevA.93.052320.
- **Programming a quantum annealer**
  - Denny Dahl. "Programming with D-Wave: Map Coloring Problem". D-Wave Systems, November 2013. URL: https://www.dwavesys.com/sites/default/files/Map%20Coloring%20WP2.pdf
  - Scott Pakin. "A Quantum Macro Assembler". 20th Annual IEEE High Performance Extreme Computing Conference (HPEC 2016), Waltham, Massachusetts, September 2016. DOI: 10.1109/HPEC.2016.7761637.
  - QMASM: https://github.com/lanl/qmasm
  - edif2qmasm: https://github.com/lanl/edif2qmasm
  - Yosys Open SYnthesis Suite: http://www.clifford.at/yosys/
  - Verilog: https://en.wikipedia.org/wiki/Verilog
- **Parting thoughts**
  - Vasil S. Denchev, Sergio Boixo, Sergei V. Isakov, Nan Ding, Ryan Babbush, Vadim Smelyanskiy, John Martinis, and Hartmut Neven. "What is the Computational Value of Finite-Range Tunneling?". Physical Review X, **6**, 031015 (2016). DOI: 10.1103/PhysRevX.6.031015.

Acknowledgments

- **Andrew Landahl (Sandia National Laboratories)**
  - For clearly explaining what is and is not known about the performance of quantum computing in general and adiabatic quantum optimization in particular and for providing a wealth of useful references
- **Yiğit Subaşı (Los Alamos National Laboratory)**
  - For spending hours with me answering my questions regarding the computation of the energy gap in an annealing process, the Kibble-Zurek mechanism, GHZ experiments, and other aspects of quantum mechanics
- **Denny Dahl (D-Wave Systems, Inc.)**
  - For numerous discussions about the D-Wave's hardware and software environment and for finding the right people at D-Wave to talk to for any issue or question
- **Trevor Lanting (D-Wave Systems, Inc.)**
  - For correcting my presentation of the physics of D-Wave's quantum processing unit
- **Murray Thom (D-Wave Systems, Inc.)**
  - For a lively email discussion about the practical computational power of different forms of quantum computers and what is and isn't accurate to claim