Programming Quantum Computers:
A Primer with IBM Q and D-Wave Exercises

by Frank Mueller, Patrick Dreher, Greg Byrd

http://moss.csc.ncsu.edu/~mueller/qc/qc-tut

North Carolina State University

NC STATE UNIVERSITY
Department of Computer Science

NC STATE UNIVERSITY
Electrical and Computer Engineering

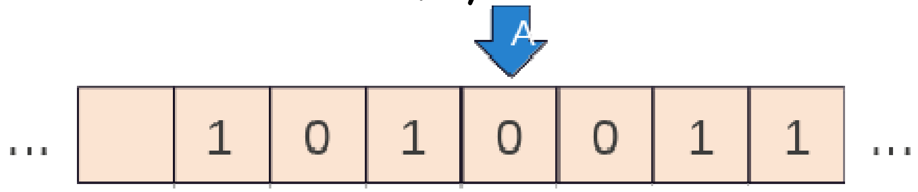
FOUNDED MARCH 7, 1887

Overview

- Welcome
- Introduction to Quantum Computing (Patrick Dreher)
  - Postulates of Quantum Mechanics, Linear Algebra, Qubits
  - Quirk Simulation
- Gate-Level Quantum Computing (Greg Byrd)
  - Quantum Gates, Circuits, and Algorithms
  - IBM Q Operation
  - IBM Q Programming with Qiskit
- Adiabatic Quantum Computing (Frank Mueller)
  - Basics of Quantum Annealing and QUBOs
  - D-Wave Programming
- Programming Exercises with IBM Q and D-Wave

What is a computer?

- Mathematical abstraction: a Turing machine
  - $M = \{Q, \Gamma, b, \Sigma, \delta, qo, F)$
    - 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”



What else is a computer?

- Nondeterministic Turing machine
  - Replace transition function with a transition relation
  - Contradictions are allowed
  - Example: "If you're in state A and you see a 0, then simultaneously (i) write a 1, move to the left, and enter state B; (ii) write a 0, move to the right, and enter state C; and (iii) write a 1, move to the right, and enter state B."
  - At each step, oracle suggests best path to take (unrealistic!)
- Quantum Turing machine
  - Same 7-tuple as in the base Turing machine
  - $M = \{Q, \Gamma, b, \Sigma, \delta, qo, F)$
  - 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

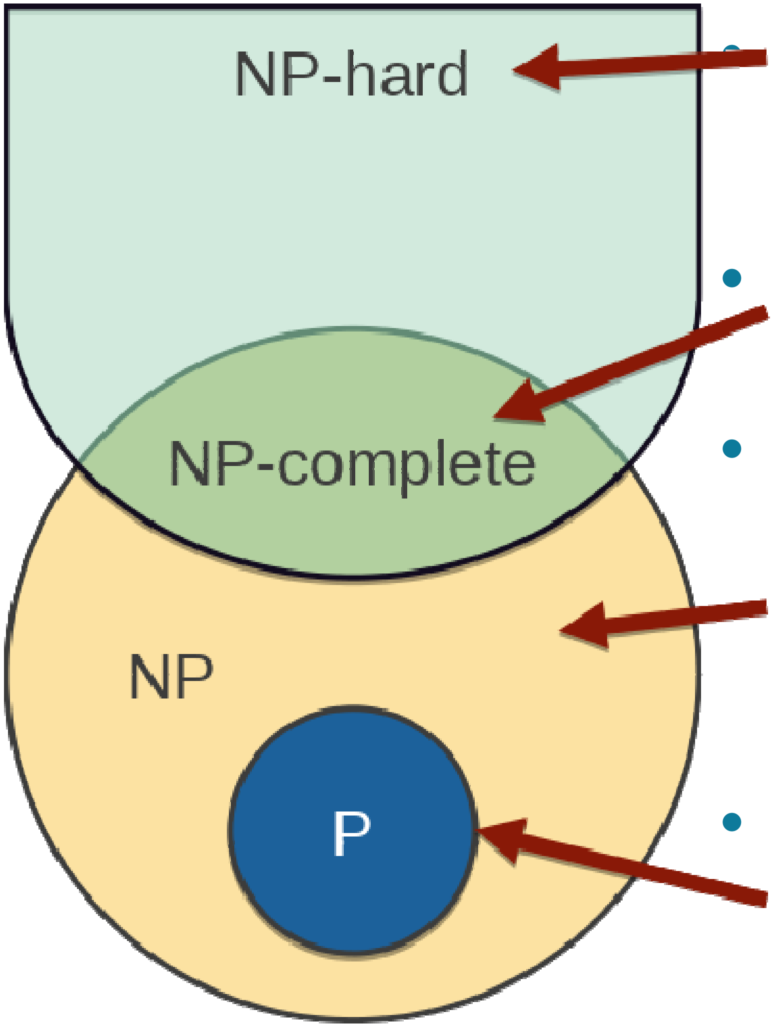
Introduction to Complexity Theory

- 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

Introduction to Complexity Theory (cont.)

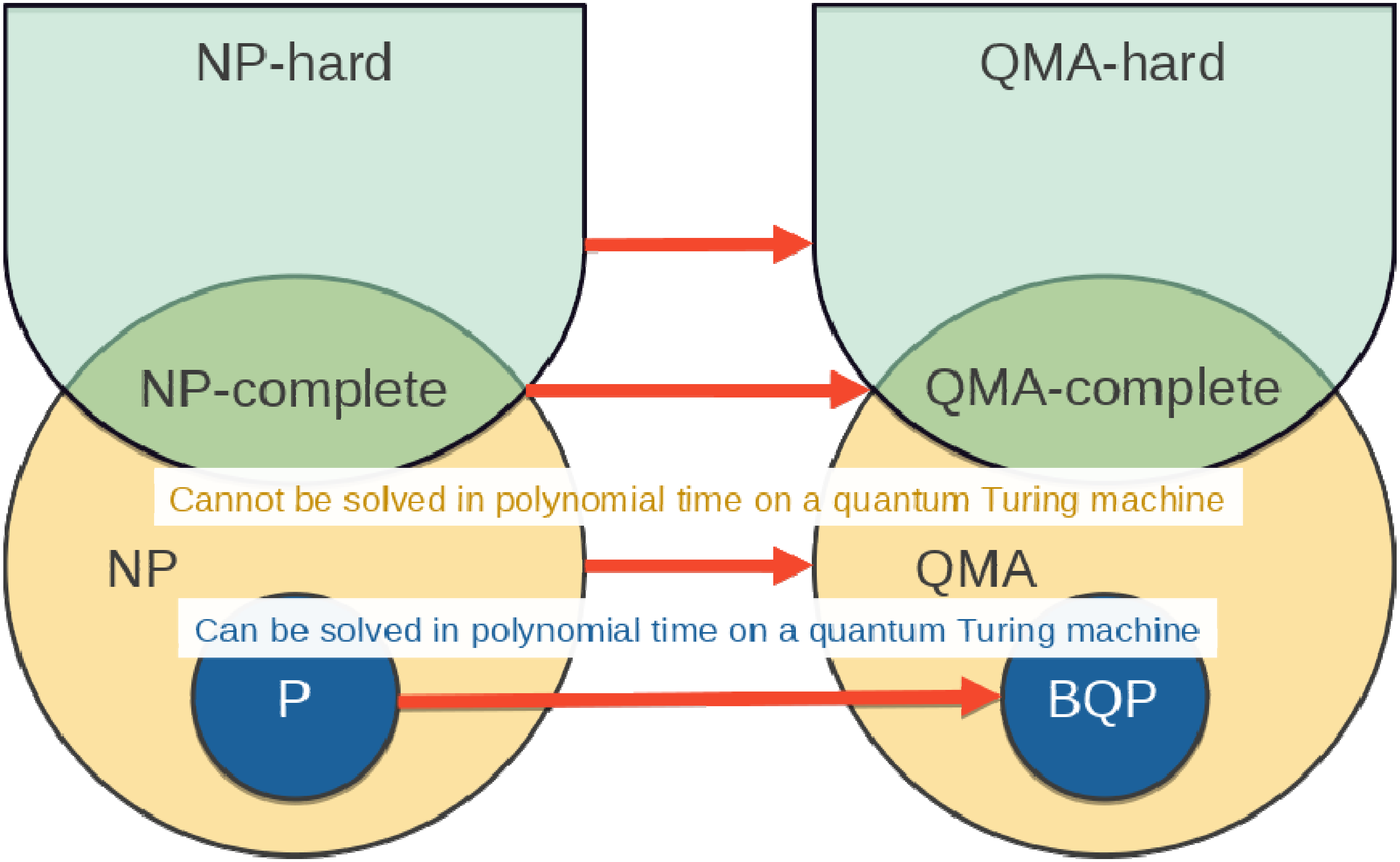
- Categorize problems into complexity classes
  - Goal: Determine which complexity classes are subsets or proper subsets of which other classes (i.e., representing, respectively, “no harder” or “easier” problems)
  - Approach is typically based on reductions: proofs that an efficient solution to a problem 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

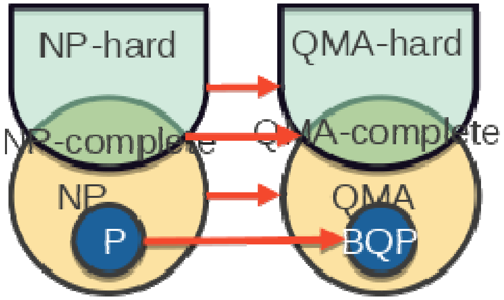


- Problems at least as hard as those in NP
  - Not necessarily decision problems
  - Ex.: Given weighted graph, find shortest-length Hamiltonian path?
- Hardest of the problems in NP
  - Ex.: Given set of integers, is there a subset whose sum is 0?
- "Hard" decision problems
  - Can be solved in polynomial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

Quantum [Merlin Arthur] (QMA) Computing Complexity Classes

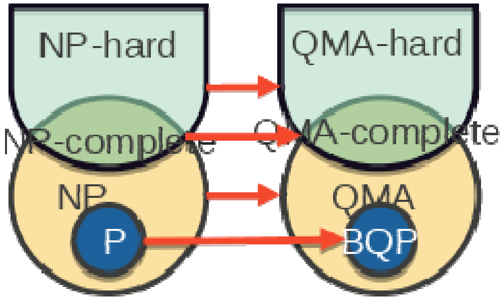


What Do We Know?



- Short answer: Almost nothing
- P vs. NP
  - P ⊆ NP
  - ??? P = NP or P ≠ NP; conjectured that P ≠ NP
- NP-intermediate vs. NP-complete
  - NP-intermediate: set of problems in NP but not in NP-complete
  - NP-intermediate ⊆ NP-complete
  - ??? NP-intermediate = NP-complete
  - 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

What Do We Know? (cont.)



- P vs. BQP
  - $P \subseteq BQP$
  - ??? $P = BQP$ or $P \neq BQP$
  - Implication: If $P = BQP$, then quantum offer no substantial (i.e., superpolynomial) performance advantage over classical
- NP-complete vs. BQP
  - ??? BQP vs. NP-complete; conjectured $BQP \subset$ NP-complete
  - Implication: Believed that quantum computers cannot solve NP-complete problems in polynomial time
- Initial focus: Quantum supremacy → break complexity class
- Today's focus: Quantum advantage → faster than classical
  - By constant factor

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)
  - $O(2^{\sqrt[3]{n}}) \rightarrow O((\log n)^3)$

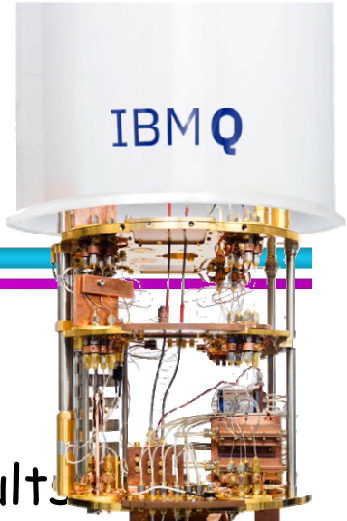
Quantum Architectures

1. Quantum annealer (D-Wave)
   - Specialized: optimization problems → find lowest energy level
   - Uses tunneling and entanglement
   - Better than classical? → unknown, maybe significant speedup
2. Approximate quantum [gate] computer (IBM Q, Regetti, IonQ…)
   - More general: optimization, quantum chemistry, machine learning
   - Superposition, entanglement
   - Better than classical? → likely, sign. Speedup: "advantage"
3. Fault-tolerant quantum computer (in "some years" from now)
   - Deals w/ errors (noise) algorithmically
   - Most general: crypto, search, and any of the above ones
   - Need 1000 physical qubits per virtual ("error-free") qubit
   - Better than classical? → proved theoretically: "supremacy"

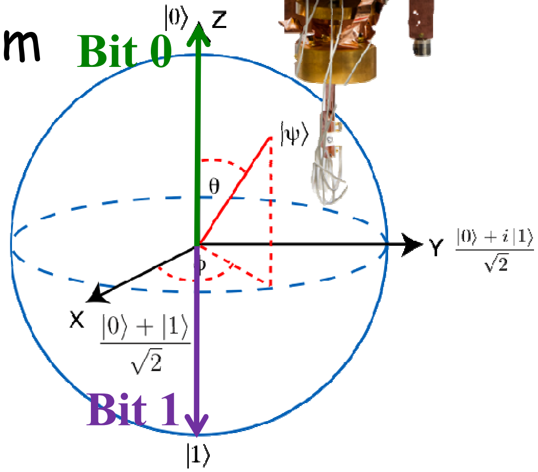
Quantum Algorithms (Gate Model)

- 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
- Google "quantum algorithm zoo" → 60 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

Gate model (cont.)



- Examples: IBM Q, Regatti, IonQ, Intel, Google…
- Programming = set parameters of physics experiment, use lasers/radio freq. to energize qubits, observe results
  - Lasers/radio freq. triggered by your program
  - Program = circuit of basic quantum gates
    - Quantum: CNOT …, classical: NAND …
    - Clock rate in us range
- 2^N states → qubits in “superposition”
  - IBM Q: 20 qubits → 2^{20} states today
  - Qubit: $|0\rangle = \begin{pmatrix}1\\0\end{pmatrix}$, $|1\rangle = \begin{pmatrix}0\\1\end{pmatrix}$ as column vector →
  - Superposition: 0 & 1 “at the same time” $|\psi\rangle = a|0\rangle + b|1\rangle$, $|a|^2 + |b|^2 = 1$
  - Example: 3 qubits, overall state $|\psi\rangle = a|000\rangle + b|001\rangle + c|010\rangle$…
    - Repeat measurement→probability per state: $|a|^2, |b|^2, |c|^2$
    - new results every few ms



Bloch sphere

Overview

- Welcome
- Introduction to Quantum Computing (Patrick Dreher)
  - Postulates of Quantum Mechanics, Linear Algebra, Qubits
  - Quirk Simulation
- Gate-Level Quantum Computing (Greg Byrd)
  - Quantum Gates, Circuits, and Algorithms
  - IBM Q Operation
  - IBM Q Programming with Qiskit
- Adiabatic Quantum Computing (Frank Mueller)
  - Basics of Quantum Annealing and QUBOs
  - D-Wave Programming
- Programming Exercises with IBM Q and D-Wave