Week 13: Quantum complexity theory

COMS 4281 (Fall 2025)

Admin

1. November 26: Pset2 Theory due Wednesday midnight.
2. November 28: No class Thanksgiving day.
3. December 2: Final review.
4. December 4: In-class final.
5. December 14: Pset2 Coding due.

Quantum algorithms, so far

- Simons Algorithm
- Order Finding/Factoring, Phase Estimation
- Grover Search, Quantum Counting, Amplitude Amplification

Quantum algorithms, so far

- Simons Algorithm
- Order Finding/Factoring, Phase Estimation
- Grover Search, Quantum Counting, Amplitude Amplification

Nearly 30 years after Shor's and Grover's algorithm, we still only have a very murky idea of when quantum computers are better than classical computers.

The formal study of this question is the focus of **quantum complexity theory**.

Computational complexity theory

Study of various **computational resources** needed to solve computational problems.

- Time
- Space
- Randomness
- Interaction
- Non-determinism
- Quantumness
- ...

Computational complexity theory

The central questions:

1. How do these computational resources relate to each other?
2. What are the tradeoffs?

Computational complexity theory

The central questions:

1. How do these computational resources relate to each other?
2. What are the tradeoffs?
3. Does non-determinism help speed up computations (P vs NP)
4. If a problem can be solved using a small amount of memory, can it also be solved using a small amount of time?
5. Can quantum computers efficiently solve problems that are hard for classical computers?

Complexity classes

Complexity classes are used to classify and compare computational problems according to the computational resources needed to solve them.

Complexity classes

Complexity classes are used to classify and compare computational problems according to the computational resources needed to solve them.

The focus is on **decision problems**, which are computational problems where for each **input** there is a **binary output** ("yes" or "no").

Complexity classes

Complexity classes are used to classify and compare computational problems according to the computational resources needed to solve them.

The focus is on **decision problems**, which are computational problems where for each **input** there is a **binary output** ("yes" or "no").

Example: Graph connectivity problem

- **Input**: graph G
- **Output**: is G connected?

Some complexity classes

P - polynomial time

Decision problems that can be solved by **deterministic** algorithms running in time $O(n^c)$ where n is the length of the input.

Traditionally considered the notion of **efficient classical computation** in theoretical computer science.

P - polynomial time

Decision problems that can be solved by **deterministic** algorithms running in time $O(n^c)$ where n is the length of the input.

Traditionally considered the notion of **efficient classical computation** in theoretical computer science.

Examples: graph connectivity, determining if a number is prime, computing shortest paths in a graph

NP - nondeterministic polynomial time

Decision problems whose solutions can be **verified** in polynomial time. If the answer to the input is "yes", then there exists a solution/certificate/proof that is efficiently checkable.

NP - nondeterministic polynomial time

Decision problems whose solutions can be **verified** in polynomial time. If the answer to the input is "yes", then there exists a solution/certificate/proof that is efficiently checkable.

Examples: traveling salesman person problem, boolean satisfiability, factoring.

Most optimization/search problems are in NP. Many are **NP-complete**, meaning they are amongst the hardest NP problems.

BPP - bounded-error probabilistic polynomial time

Decision problems that can be solved by a **randomized**, polynomial time algorithm.

The correct answer must be obtained with high probability (say 99%).

BPP - bounded-error probabilistic polynomial time

Decision problems that can be solved by a **randomized**, polynomial time algorithm.

The correct answer must be obtained with high probability (say 99%).

Examples: any problem in P, polynomial identity testing

It is conjectured that P = BPP (i.e. randomization does not help for efficient computation).

PSPACE - polynomial space

Decision problems that can be solved by a deterministic algorithm that uses $O(n^c)$ bits of space where n is the input length.

Captures the notion of problems that can be solved using a small amount of **memory**.

PSPACE - polynomial space

Decision problems that can be solved by a deterministic algorithm that uses $O(n^c)$ bits of space where n is the input length.

Captures the notion of problems that can be solved using a small amount of **memory**.

Examples: all of P, NP, BPP. Generalized tic-tac-toe, Super Mario Bros.

EXP - exponential time

Decision problems that can be solved by a deterministic algorithm that runs in $O(2^{n^c})$ time.

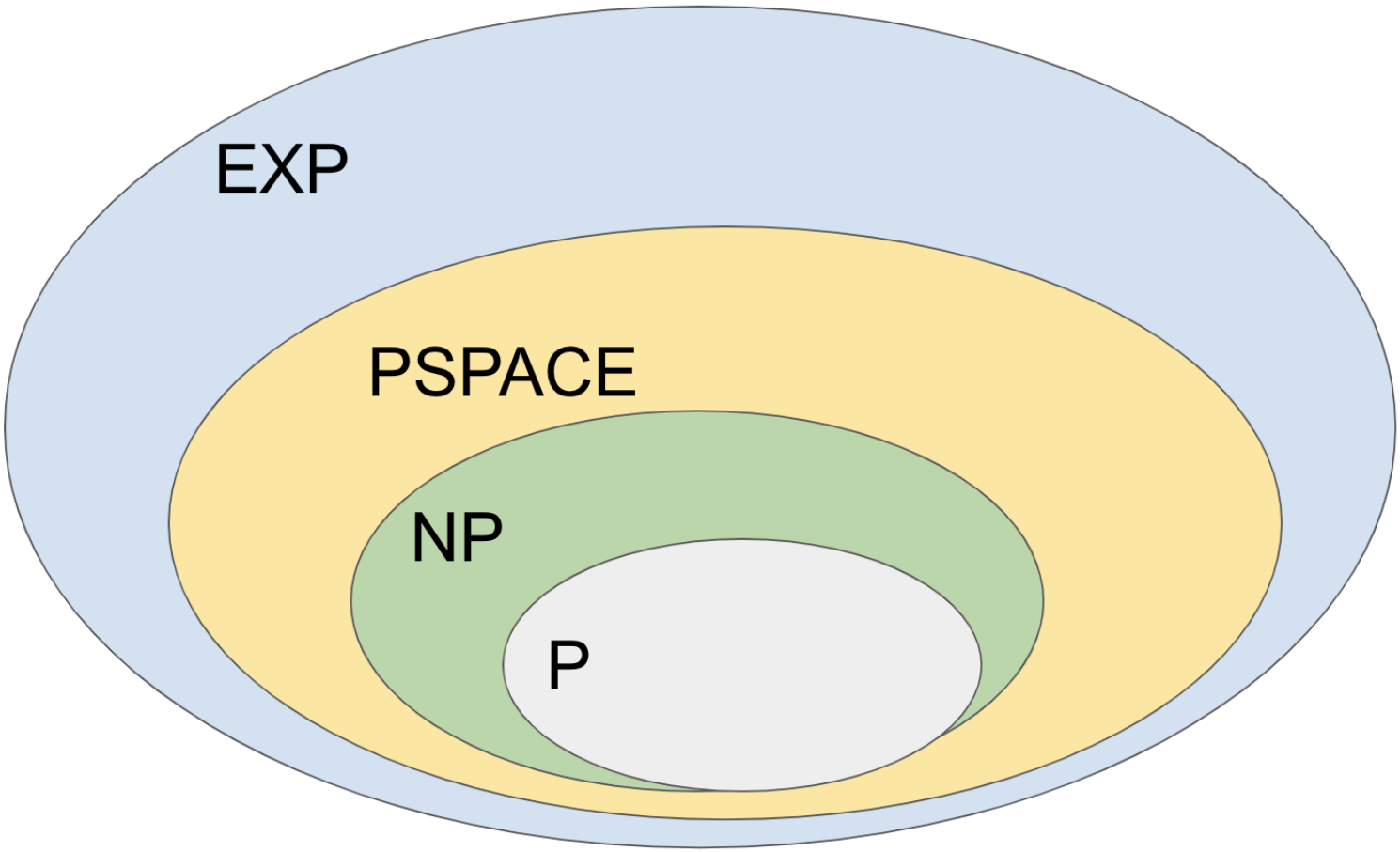
EXP - exponential time

Decision problems that can be solved by a deterministic algorithm that runs in $O(2^{n^c})$ time.

Examples: all of PSPACE, generalized Chess.

Classical complexity classes

The only separation we know is P $\neq$ EXP.



BQP - bounded-error quantum polynomial time

Decision problems that can be solved by a **quantum** algorithm (i.e. a quantum circuit) of $O(n^c)$ size with high probability.

Captures the notion of **efficient quantum computation**.

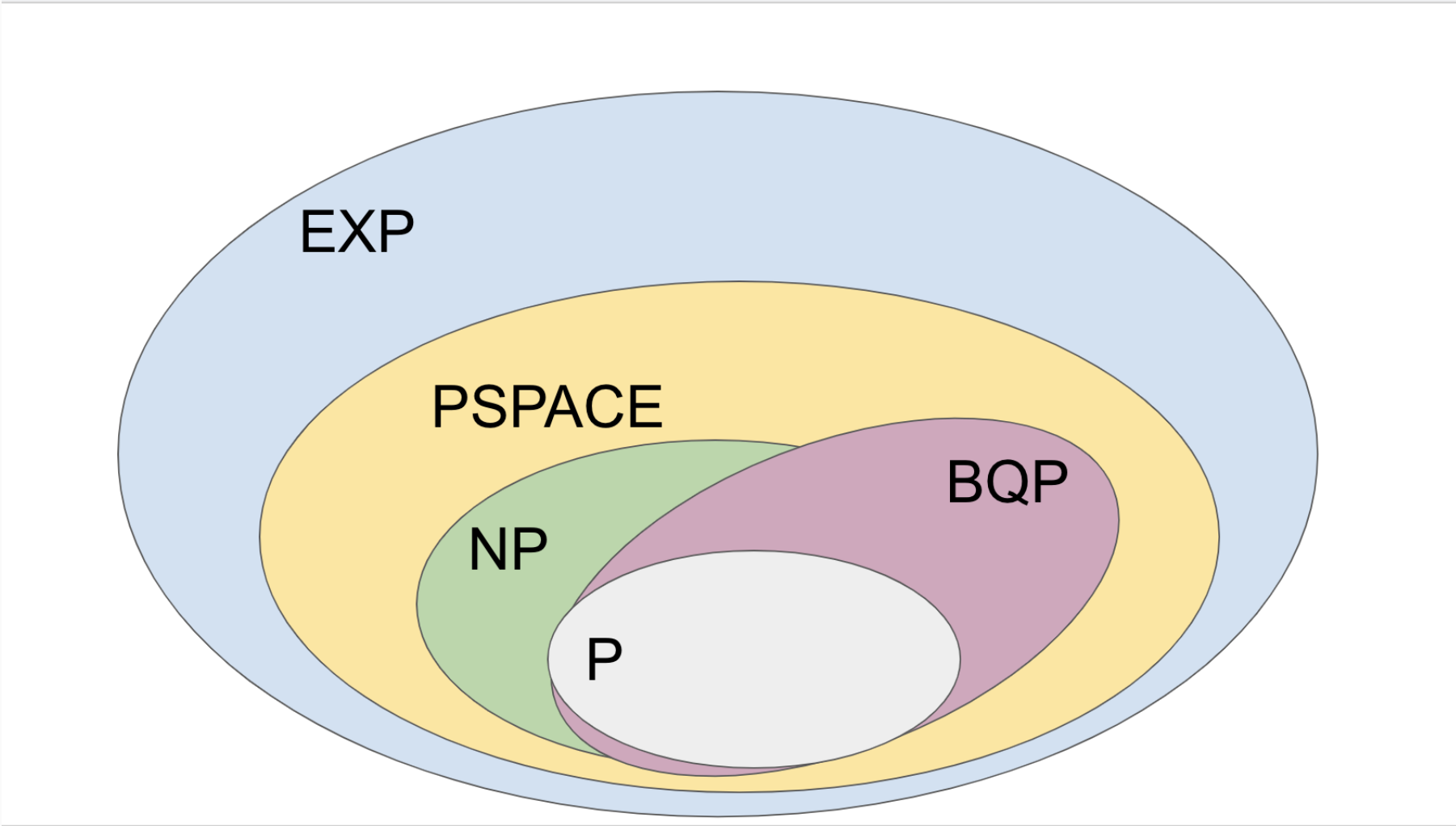
BQP - bounded-error quantum polynomial time

Decision problems that can be solved by a **quantum** algorithm (i.e. a quantum circuit) of $O(n^c)$ size with high probability.

Captures the notion of **efficient quantum computation**.

Examples: all of P, BPP, factoring, simulating quantum physics.

Where does BQP live?



The central questions

BQP $\subseteq$ **BPP**?

In other words, is there an efficient classical simulation of quantum computation?

The central questions

BQP $\subseteq$ **BPP**?

In other words, is there an efficient classical simulation of quantum computation?

NP $\subseteq$ **BQP**?

Can quantum computers be used to solve hard optimization problems like SAT or Traveling Salesperson Problem?

The central questions

BQP $\subseteq$ **BPP**?

In other words, is there an efficient classical simulation of quantum computation?

NP $\subseteq$ **BQP**?

Can quantum computers be used to solve hard optimization problems like SAT or Traveling Salesperson Problem?

What are classical upper bounds on BQP?

What resources does a classical computer need in order to simulate quantum computations?

Classical simulations of quantum computations

Different notions of simulation

Let C be a quantum circuit on n qubits and T single- and two-qubit gates. Initial state $|0\rangle^{\otimes n}$. Three different types of classical simulation of C:

Different notions of simulation

Let C be a quantum circuit on n qubits and T single- and two-qubit gates. Initial state $|0\rangle^{\otimes n}$. Three different types of classical simulation of C:

1. **State-vector simulation**: compute a classical description of the state $C|0\rangle^{\otimes n}$.

Different notions of simulation

Let C be a quantum circuit on n qubits and T single- and two-qubit gates. Initial state $|0\rangle^{\otimes n}$. Three different types of classical simulation of C:

1. **State-vector simulation**: compute a classical description of the state $C|0\rangle^{\otimes n}$.
2. **Amplitude estimation**: output an estimate of the amplitude of a particular basis state $|x\rangle$ in the state $C|0\rangle^{\otimes n}$.

Different notions of simulation

Let C be a quantum circuit on n qubits and T single- and two-qubit gates. Initial state $|0\rangle^{\otimes n}$. Three different types of classical simulation of C:

1. **State-vector simulation**: compute a classical description of the state $C|0\rangle^{\otimes n}$.
2. **Amplitude estimation**: output an estimate of the amplitude of a particular basis state $|x\rangle$ in the state $C|0\rangle^{\otimes n}$.
3. **Sampling**: output a string $x \in \{0,1\}^n$ sampled as if the state $C|0\rangle^{\otimes n}$ were measured in the standard basis.

State-vector simulation

Let the gates of C be $G_1, G_2, \ldots, G_T$. To compute a classical description of the state $C|0\rangle^{\otimes n}$, we necessarily need at least 2^n time, just to write down the entire description of the vector.

State-vector simulation

Let the gates of C be $G_1, G_2, \ldots, G_T$. To compute a classical description of the state $C|0\rangle^{\otimes n}$, we necessarily need at least 2^n time, just to write down the entire description of the vector.

1. Initialize vector $|\psi_0\rangle = |0\rangle^{\otimes n}$ in classical memory.

State-vector simulation

Let the gates of C be $G_1, G_2, \ldots, G_T$. To compute a classical description of the state $C|0\rangle^{\otimes n}$, we necessarily need at least 2^n time, just to write down the entire description of the vector.

1. Initialize vector $|\psi_0\rangle = |0\rangle^{\otimes n}$ in classical memory.
2. For $t = 1, \ldots, T$:
   2.1 Compute vector $|\psi_t\rangle = G_t |\psi_{t-1}\rangle$.

State-vector simulation

Let the gates of C be $G_1, G_2, \ldots, G_T$. To compute a classical description of the state $C|0\rangle^{\otimes n}$, we necessarily need at least 2^n time, just to write down the entire description of the vector.

1. Initialize vector $|\psi_0\rangle = |0\rangle^{\otimes n}$ in classical memory.
2. For $t = 1, \ldots, T$:
   2.1 Compute vector $|\psi_t\rangle = G_t |\psi_{t-1}\rangle$.
3. Output vector $|\psi_T\rangle$.

State-vector simulation

Let the gates of C be $G_1, G_2, \ldots, G_T$. To compute a classical description of the state $C|0\rangle^{\otimes n}$, we necessarily need at least 2^n time, just to write down the entire description of the vector.

1. Initialize vector $|\psi_0\rangle = |0\rangle^{\otimes n}$ in classical memory.
2. For $t = 1, \ldots, T$:
   2.1 Compute vector $|\psi_t\rangle = G_t |\psi_{t-1}\rangle$.
3. Output vector $|\psi_T\rangle$.

Space complexity: $O(2^n)$ – **exponential space**

State-vector simulation

Let the gates of C be $G_1, G_2, \ldots, G_T$. To compute a classical description of the state $C|0\rangle^{\otimes n}$, we necessarily need at least 2^n time, just to write down the entire description of the vector.

1. Initialize vector $|\psi_0\rangle = |0\rangle^{\otimes n}$ in classical memory.
2. For $t = 1, \ldots, T$:
   2.1 Compute vector $|\psi_t\rangle = G_t |\psi_{t-1}\rangle$.
3. Output vector $|\psi_T\rangle$.

Space complexity: $O(2^n)$ – **exponential space**
Time complexity: $O(T \cdot 2^n)$ – **exponential time**.

Amplitude estimation

What if we just wanted to estimate a particular **amplitude** of the final state $C|0\rangle^{\otimes n}$, for example the amplitude on all the all zeroes string?

Amplitude estimation

What if we just wanted to estimate a particular **amplitude** of the final state $C|0\rangle^{\otimes n}$, for example the amplitude on all the all zeroes string?

Can we do this without storing the entire state vector in memory?

Amplitude estimation

What if we just wanted to estimate a particular **amplitude** of the final state $C|0\rangle^{\otimes n}$, for example the amplitude on all the all zeroes string?

Can we do this without storing the entire state vector in memory?

Yes! This can be computed in **polynomial space**.

Amplitude estimation

Suppose we want to compute the amplitude of the basis state $|x\rangle$ in $|\psi_T\rangle$.

Amplitude estimation

Suppose we want to compute the amplitude of the basis state $|x\rangle$ in $|\psi_T\rangle$.

The final state $|\psi_T\rangle$ is equal to

$$G_T G_{T-1} \cdots G_2 G_1 |0\rangle^{\otimes n}.$$

Amplitude estimation

Suppose we want to compute the amplitude of the basis state $|x\rangle$ in $|\psi_T\rangle$.

The final state $|\psi_T\rangle$ is equal to

$$G_T G_{T-1} \cdots G_2 G_1 |0\rangle^{\otimes n}.$$

The amplitude we want to compute is equal to

$$\langle x|\psi_T\rangle = \langle x| G_T G_{T-1} \cdots G_2 G_1 |0\rangle^{\otimes n}.$$

Amplitude estimation

We can insert identity matrices in between each gate:

$$\langle x| \, G_T \cdot \mathrm{I} \cdot G_{T-1} \cdots G_2 \cdot \mathrm{I} \cdot G_1 \, |0\rangle^{\otimes n} .$$

Amplitude estimation

We can insert identity matrices in between each gate:

$$\langle x| G_T \cdot \mathrm{I} \cdot G_{T-1} \cdots G_2 \cdot \mathrm{I} \cdot G_1 |0\rangle^{\otimes n}.$$

Substituting each identity with the equation $\mathrm{I} = \sum_{y \in \{0,1\}^n} |y\rangle \langle y|$, we get

$$\langle x| G_T \cdot \Big(\sum_{y_{T-1}} |y_{T-1}\rangle \langle y_{T-1}| \Big) \cdot G_{T-1} \cdots G_2 \cdot \Big(\sum_{y_1} |y_1\rangle \langle y_1| \Big) \cdot G_1 |0\rangle^{\otimes n}.$$

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

The summation is over all possible combinations of n-bit strings $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$, called **Feynman paths**.

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

The summation is over all possible combinations of n-bit strings $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$, called **Feynman paths**.

Each term in the sum is the product of **transition amplitudes**

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

Fix a **Feynman path** $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$.

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

Fix a **Feynman path** $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$.

Fix a transition amplitude $\langle y_t| G_t |y_{t-1}\rangle$. Given the strings y_{t-1}, y_t, this is a *number* that can be computed in $\mathrm{poly}(n)$ time,

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

Fix a **Feynman path** $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$.

Fix a transition amplitude $\langle y_t| G_t |y_{t-1}\rangle$. Given the strings y_{t-1}, y_t, this is a *number* that can be computed in $\mathrm{poly}(n)$ time, because G_t acts nontrivially on at most 2 qubits.

Amplitude estimation

For example, suppose G_t acts nontrivially on the first two qubits, and identity elsewhere. Let $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n)$ be bitstrings. Then

$$\langle a| G_t |b\rangle = \langle a_1, a_2| G_t |b_1, b_2\rangle \cdot \langle a_3, \ldots, a_n \mid b_3, \ldots, b_n\rangle .$$

Amplitude estimation

For example, suppose G_t acts nontrivially on the first two qubits, and identity elsewhere. Let $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n)$ be bitstrings. Then

$$\langle a| G_t |b\rangle = \langle a_1, a_2| G_t |b_1, b_2\rangle \cdot \langle a_3, \ldots, a_n \mid b_3, \ldots, b_n\rangle .$$

The first factor $\langle a_1, a_2| G_t |b_1, b_2\rangle$ can be computed in $O(1)$ time.

Amplitude estimation

For example, suppose G_t acts nontrivially on the first two qubits, and identity elsewhere. Let $a = (a_1, \ldots, a_n), b = (b_1, \ldots, b_n)$ be bitstrings. Then

$$\langle a| G_t |b\rangle = \langle a_1, a_2| G_t |b_1, b_2\rangle \cdot \langle a_3, \ldots, a_n \mid b_3, \ldots, b_n\rangle .$$

The first factor $\langle a_1, a_2| G_t |b_1, b_2\rangle$ can be computed in $O(1)$ time. The second factor $\langle a_3, \ldots, a_n \mid b_3, \ldots, b_n\rangle$ is 0 unless $a_3 = b_3, \ldots, a_n = b_n$.

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

For a fixed **Feynman path** $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$, the product of transition amplitudes $\langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n}$ can be computed in $\mathrm{poly}(n, T)$ time.

Amplitude estimation

Expanding all the parentheses, the amplitude is equal to

$$\sum_{y_1, y_2, \ldots, y_{T-1}} \langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n} .$$

For a fixed **Feynman path** $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$, the product of transition amplitudes $\langle x| G_T |y_{T-1}\rangle \langle y_{T-1}| G_{T-1} |y_{T-2}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n}$ can be computed in $\mathrm{poly}(n, T)$ time.

To compute the **full sum**, it suffices to have an variable for the accumulating sum, and counters keeping track of which Feynman path we're computing.

Amplitude estimation

Sum-over-histories algorithm:

1. sum $\leftarrow 0$

Amplitude estimation

Sum-over-histories algorithm:

1. sum $\leftarrow 0$
2. For $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$:
   2.1 sum $\leftarrow$ sum $+ \langle x| G_T |y_{T-1}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n}$

Amplitude estimation

Sum-over-histories algorithm:

1. sum $\leftarrow 0$
2. For $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$:
   2.1 sum $\leftarrow$ sum $+ \langle x| G_T |y_{T-1}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n}$
3. Output sum

Amplitude estimation

Sum-over-histories algorithm:

1. sum $\leftarrow 0$
2. For $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$:
   2.1 sum $\leftarrow$ sum $+ \langle x| G_T |y_{T-1}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n}$
3. Output sum

Space complexity: $O(Tn)$ – **polynomial space**.

Amplitude estimation

Sum-over-histories algorithm:

1. sum $\leftarrow 0$
2. For $y_1, y_2, \ldots, y_{T-1} \in \{0,1\}^n$:
   2.1 sum $\leftarrow$ sum $+ \langle x| G_T |y_{T-1}\rangle \cdots \langle y_2| G_2 |y_2\rangle \langle y_1| G_1 |0\rangle^{\otimes n}$
3. Output sum

Space complexity: $O(Tn)$ – **polynomial space**.
Time complexity: $2^{O(Tn)} \cdot \mathrm{poly}(n, T)$ – **exponential time**.

Sampling

Suppose we instead want to **sample** from the output distribution of the circuit C given all zeroes input.

Sampling

Suppose we instead want to **sample** from the output distribution of the circuit C given all zeroes input.

In other words, we want a probabilistic classical algorithm that outputs a string x sampled as if we measured the state $C|0\rangle^{\otimes n}$.

By using the state vector simulation approach, we can do this using exponential space and exponential time.

Sampling

Suppose we instead want to **sample** from the output distribution of the circuit C given all zeroes input.

In other words, we want a probabilistic classical algorithm that outputs a string x sampled as if we measured the state $C|0\rangle^{\otimes n}$.

By using the state vector simulation approach, we can do this using exponential space and exponential time.

We can also do this in polynomial space, by sampling each bit of x at a time.

Sampling

1. Using sum-over-histories approach to compute $p_1 = \Pr[x_1 = 1]$.

Sampling

1. Using sum-over-histories approach to compute $p_1 = \Pr[x_1 = 1]$.
2. Sample a bit x_1 that is equal to 1 with probability p_1.

Sampling

1. Using sum-over-histories approach to compute $p_1 = \Pr[x_1 = 1]$.
2. Sample a bit x_1 that is equal to 1 with probability p_1.
3. Compute $p_2 = \Pr[x_2 = 1 \mid x_1]$.

Sampling

1. Using sum-over-histories approach to compute $p_1 = \Pr[x_1 = 1]$.
2. Sample a bit x_1 that is equal to 1 with probability p_1.
3. Compute $p_2 = \Pr[x_2 = 1 \mid x_1]$.
4. Sample a bit x_2 that is equal to 1 with probability p_2.
5. $\ldots$

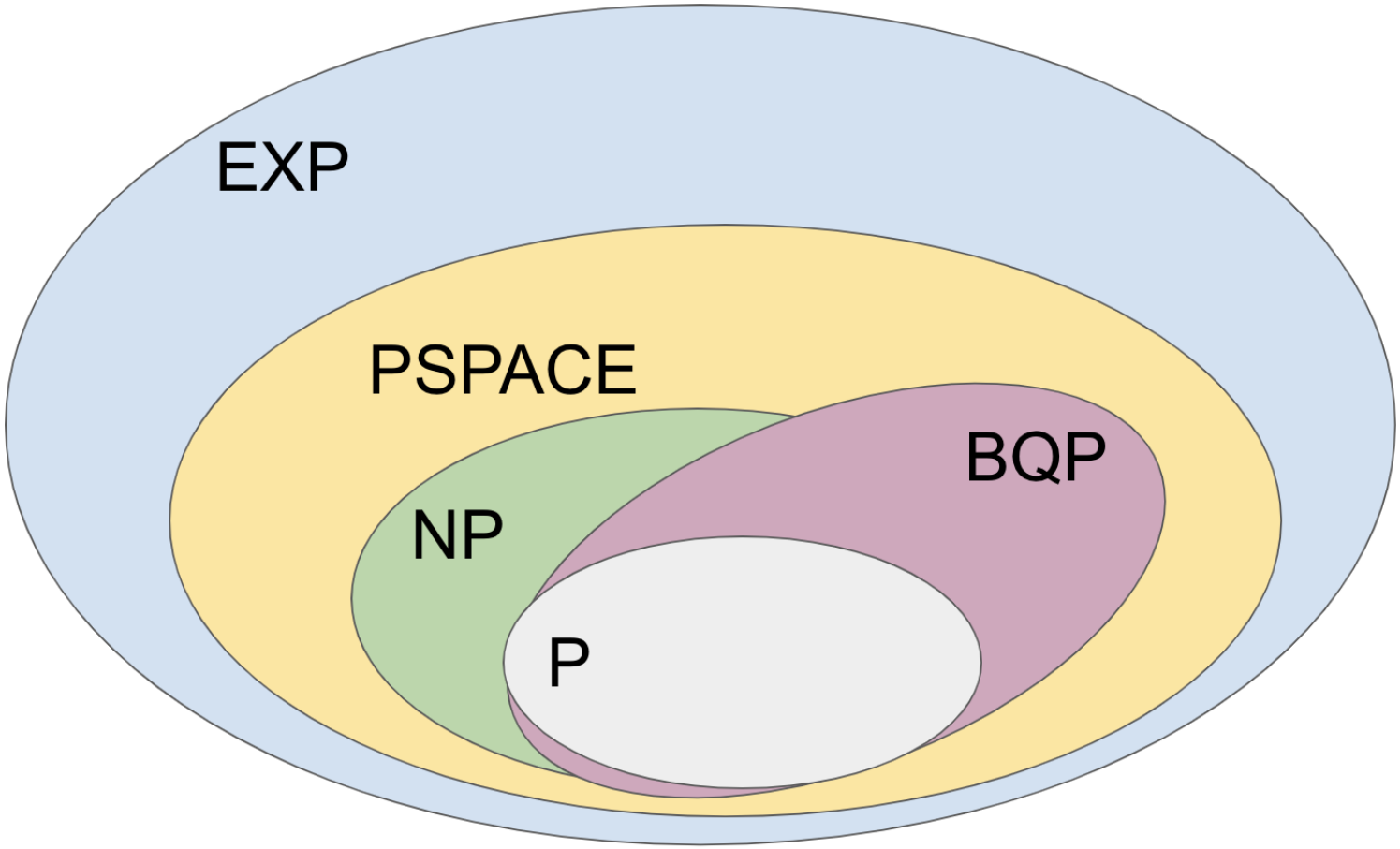
Sampling

1. Using sum-over-histories approach to compute $p_1 = \Pr[x_1 = 1]$.
2. Sample a bit x_1 that is equal to 1 with probability p_1.
3. Compute $p_2 = \Pr[x_2 = 1 \mid x_1]$.
4. Sample a bit x_2 that is equal to 1 with probability p_2.
5. $\ldots$

5. Sample a bit x_n that is equal to 1 with probability $\Pr[x_n = 1 \mid x_1, x_2, \ldots]$.

BQP vs NP

BQP vs NP?



NP

Decision problems where the “yes” inputs have solutions that can be efficiently checked.

NP

Decision problems where the "yes" inputs have solutions that can be efficiently checked.

Example: Traveling Salesman Problem

Input: Graph G, integer k

Output: Does G have a tour of length at most k that visits every city once and returns to origin?

TSP is in NP because given a proposed tour, one can check whether it visits every city once, returns to origin, and has length at most k.

NP - Equivalent Definition

A decision problem has an NP "algorithm" if the algorithm can *nondeterministically* guess a solution (if it exists), and check the solution in polynomial time. If there is no solution, no guess will be accepted.

A common misconception

Quantum computers solve problems by trying all possible solutions simultaneously and outputting a correct one!

A common misconception

Quantum computers solve problems by trying all possible solutions simultaneously and outputting a correct one!

If that were true, then NP complete problems would be instantly solvable on quantum computers.

However this simplistic picture of quantum computing is not true. The truth is much more interesting...

Evidence that NP $\not\subseteq$ BQP

We don't have the tools available to outright prove that BQP cannot solve NP-complete problems. But we can try to give evidence for this.

Evidence that NP $\not\subseteq$ BQP

We don't have the tools available to outright prove that BQP cannot solve NP-complete problems. But we can try to give evidence for this.

Bennett, Bernstein, Brassard, and Vazirani: "Strengths and weaknesses of quantum computing" (1997). Quantum computers cannot solve unstructured search problems better than quadratically faster than a classical computer.

Evidence that NP $\not\subseteq$ BQP

We don't have the tools available to outright prove that BQP cannot solve NP-complete problems. But we can try to give evidence for this.

Bennett, Bernstein, Brassard, and Vazirani: "Strengths and weaknesses of quantum computing" (1997). Quantum computers cannot solve unstructured search problems better than quadratically faster than a classical computer.

In other words, the $O(\sqrt{N})$ run time of Grover is optimal if we know nothing about the search problem.

Quantum advantage

We're still far away from being able to run Grover's algorithm or Shor's factoring algorithm. How to demonstrate quantum advantage in the near term?

Quantum supremacy task

We wish to find a computational task T that:

1. NISQ (Noisy Intermediate-Scale Quantum) machine can run T in, e.g. < 1 second.
2. Verifiable on a classical computer in a reasonable amount of time (e.g. several weeks on a supercomputing cluster)
3. Some complexity evidence that T cannot be efficiently solved by classical computers.

Quantum supremacy task

We wish to find a computational task T that:

1. NISQ (Noisy Intermediate-Scale Quantum) machine can run T in, e.g. < 1 second.
2. Verifiable on a classical computer in a reasonable amount of time (e.g. several weeks on a supercomputing cluster)
3. Some complexity evidence that T cannot be efficiently solved by classical computers.

Such a task T would demonstrate the supremacy of quantum computers over classical computers.

Quantum supremacy task

This computational task T can only happen in a "sweet spot" with $\sim$ 50-100 qubits.

Enough that it's not easy for classical computers, but not too much so that we can run it on our existing quantum computers, and also verify it using $\sim 2^{50}$ operations on a classical computer.

Random circuit sampling

Number of qubits $n = 50$
Number of gates $m = 200$
Number of samples $T =$ several million

1. Pick a random quantum circuit C acting on n qubits, with m gates.
2. Using quantum computer run circuit C on $|0 \cdots 0\rangle$, generate samples $x_1, \ldots, x_T$ from the distribution $\mathcal{D}_C$.
3. Output $x_1, \ldots, x_T$.

Random circuit sampling

Number of qubits $n = 50$
Number of gates $m = 200$
Number of samples $T =$ several million

1. Pick a random quantum circuit C acting on n qubits, with m gates.
2. Using quantum computer run circuit C on $|0 \cdots 0\rangle$, generate samples $x_1, \ldots, x_T$ from the distribution $\mathcal{D}_C$.
3. Output $x_1, \ldots, x_T$.

The distribution $\mathcal{D}_C$: each sample x occurs with probability $p_C(x) = |\langle x| C |0 \cdots 0\rangle|^2$.

Hardness of random circuit sampling?

A quantum computer, by definition, can easily perform the random circuit sampling task. How hard is it for a classical computer to do the same?

Hardness of random circuit sampling?

A quantum computer, by definition, can easily perform the random circuit sampling task. How hard is it for a classical computer to do the same?

Theorem (Bouland, Fefferman, Nirkhe, Vazirani 2019): There is no classical algorithm that, given circuit C, can generate samples from $\mathcal{D}_C$ with high probability, unless the polynomial hierarchy collapses to the third level.

Hardness of random circuit sampling?

"X unless the polynomial hierarchy collapses" is complexity theory evidence that X is true. It is a generalization of the assumption that $P \neq NP$.

Hardness of random circuit sampling?

"X unless the polynomial hierarchy collapses" is complexity theory evidence that X is true. It is a generalization of the assumption that $P \neq NP$.

This theorem talks about sampling *exactly* from $\mathcal{D}_C$. However, not even the quantum computer can do that, because it has some small amount of noise. One can ask how hard is it to *approximately sample* from $\mathcal{D}_C$. It is conjectured that this is still hard for classical algorithms (assuming polynomial hierarchy does not collapse).

Verification of random circuit sampling

How do you check whether a bunch of samples $x_1, \ldots, x_T$ were generated by $\mathcal{D}_C$? There is no known efficient way of doing so.

Verification of random circuit sampling

How do you check whether a bunch of samples $x_1, \ldots, x_T$ were generated by $\mathcal{D}_C$? There is no known efficient way of doing so.

Idea: Heavy output generation (HOG) test

1. Use a classical supercomputer to compute the median α of all $p_C(x)$.
2. If at least $2/3$ of $x_1, \ldots, x_T$ are **heavy** (meaning $p_C(x_i) \geq \alpha$), then accept. Otherwise reject.

Verification of random circuit sampling

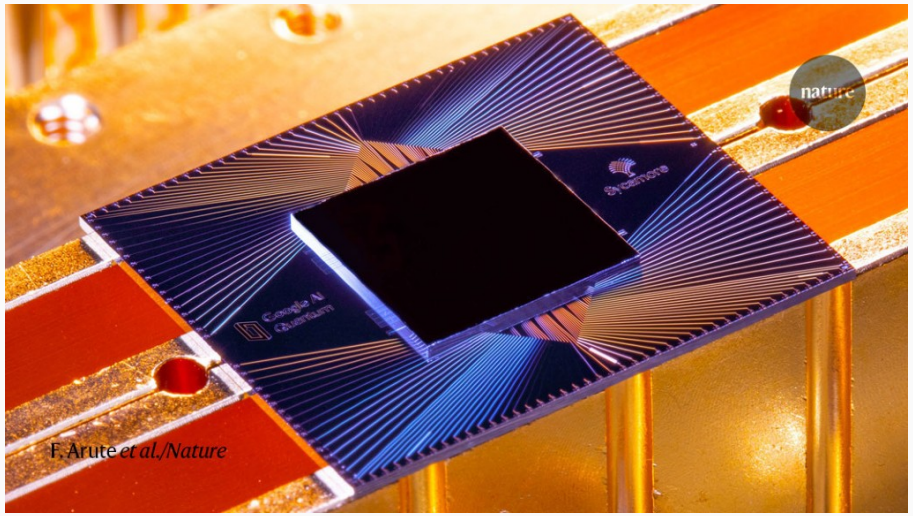
Intuition: heavy x's form the bulk of the probability mass of distribution $\mathcal{D}_C$. Sampling from $\mathcal{D}_C$ should yield a lot of heavy strings.

However, it should also be hard for a classical computer to predict, given a circuit C and x, whether $p_C(x)$ is above the median.

If a quantum machine is able to consistently output heavy strings from $\mathcal{D}_C$, then the machine must've "done something right".

Experimental demonstrations

Conducted in Fall 2019. Ran many circuits on their 49-qubit "Sycamore" chip.



Extremely noisy: signal-to-noise ratio is about 1%. (However, Google claims it is enough to verify the sampling).

Recently: many challenges to this claim (faster classical simulations, noisy sampling may not be as hard as we thought).