Quantum Complexity Theory

Eyad Loutfi

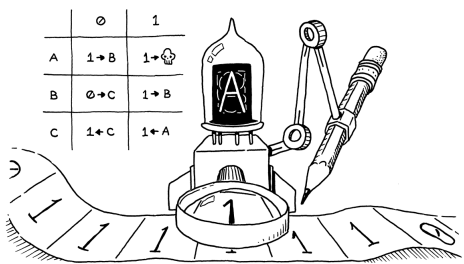
"My contention is the following: Quantum mechanics is what you would inevitably come up with if you started from probability theory, and then said, let's try to generalize it so that the numbers we used to call 'probabilities' can be negative numbers. As such, the theory could have been invented by mathematicians in the 19th century without any input from experiment. It wasn't, but it could have been."
- Scott Aaronson

Section 1

Basics of Complexity

Turing Machines

- First we need to introduce our model of computation. The classic model of computation is the *Turing machine*.
- TM's are defined over a finite alphabet, have an infinite tape that has the input at the start of it, a read head with internal state that starts at the start of the tape and can write or move left or right all according to some finite instructions (represented as a transition function based on the head's state and character in the cell it's over).



Turing Machines

- A more intuitive way to think about TM's - and that makes it clear why this is a very natural definition to come up with if you live in a time before computers - is to consider a model with the following features:
  - The world as an idealized grid
  - An idealized mathematician who can only move one cell at a time
  - Can read and write one symbol at a time (alternatively, can simply change the symbols/"alphabet" to represent words, doesn't matter so long as there's only a finite number of choices).
- All choices one can make is finite - implicitly a function or "response" of a finite number of states of ourselves and our surrounding environment. Note these "finite instructions" can still yield unbounded behavior. We're allowing behavior to go on forever, whereas in real life us and computers have finite lives.

Turing Machines

- While this might seem extremely broad and philosophical - and not specifically the same: That's the point! While we won't prove it, the Turing Machine model is very robust, you can change numerous specifics without effecting most results in the field.
- i.e. changing the number of tapes or the dimension of the tape does not change computability at all.
- Moreover if we're trying to study problems based on intrinsic difficulty, complexity theorists often classify problems broadly on whether they require polynomial or exponential time. These changes to a TM can provably gain at most only a polynomial speedup, so it does not make a difference. This all points in the direction of the Turing machine being the "right model."

Computability

- Specifically, the *Church Turing* thesis states that anything that can be "computed" can be done by a Turing machine. While this is not a formal statement, and therefore not the type that can even be proved, hopefully you've seen why intuitively it feels self evident.
- Note that once we fix the exact TM model, like programs they end up being defined by precise descriptions, so in a sense they are strings themselves!
- One can devise an encoding scheme of all programs – many such encodings exist, could use all binary strings or all ASCII strings, etc.

Computability

- More importantly, the encoding is done in such a way that a program can recognize it – such that we can define a program to simulate the behavior of a machine as a result of parsing the encoding.
- This means we can define a program to be able to simulate any other program – a very important result!
- Other important definitions: A set is “recursively enumerable,” or RE if there exists a program that can give a “yes” answer to any string in the set. A set is “computable” or “decidable" if a program like that exists that also gives a “no” answer for any string not in the set.
- These properties are not guaranteed in general due to possibility of infinite looping.

Complexity Basics

- Computability and complexity theory are the study of the fundamental limitations of computation - namely computability in the former, and time and space requirements in the latter.
- Specifically we will often study these in the context of *decision problems*, problems phrased as "yes" or "no" answers to whether an input is in a set. Typically, the decision problem and the corresponding set - or "language" that defines it are used interchangeably in the language of theoretical computer science.

Uncomputability

We can further prove not all decision problems are computable.

Theorem

The halting problem is undecidable. That is - the set of all pairs of encodings of Turing Machines with inputs (so an arbitrary $\langle M, x \rangle$), such that that respective Turing machine halts on that input - is an uncomputable set.

Proof

1. Suppose not. That is $\exists$ TM M that decides this set.
2. We can create a new machine M' that simulates M on the input to M', such that if the input is an encoding of a machine that halts, we program M' to loop forever, while if it's a machine that loops forever, we program M' to halt.
3. Now if we give M' the input $\langle M', \langle M' \rangle \rangle$, this implies M' halts on $\langle M' \rangle \Leftrightarrow M'$ loops forever on it - a contradiction.

Diagonalization

- This technique of creating a contradiction through self reference, called diagonalization will be seen again.
- Often involves assuming some self referential structure with a property is true, then constructing a similar structure that fulfills the opposite of that property, asking a carefully constructed question about itself that will lead to contradiction.
- Diagonalization was the *cornerstone* of basically all early results in logic, set theory, computability, and complexity theory during the late 19th and first half of the 20th century.

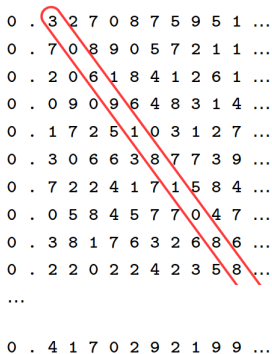
Diagonalization - Other Examples

- "This sentence is a lie." Is this sentence the truth or a lie?
- Descartes - "I think therefore I am." Suppose your thoughts do not exist. Now ask "does the thought of 'your thoughts do not exist'" exist?
- Russell - Suppose the set of all sets that do not contain themselves exists. Now ask, "does this set contain itself"?
- Godel - Consider the guaranteed to exist Godel sentence of any axiom system F that contains even just basic arithmetic, which is the statement $G \Leftrightarrow$ "F does not prove G". Now ask, "can F prove G"?

Σ

Diagonalization - Other Examples

- Cantor - Suppose the real numbers between 0 and 1 can be listed on a list enumerated by the natural numbers. Is the real number in $(0, 1)$ that's ith digit is different from the ith digit of the ith real number on that list, in that list? (This is why it's "diagonal").



Complexity Basics

- P is the class of all computable sets for which a Turing machine exists that gives a "yes" or "no" answer to membership with a polynomial number of steps relative to the input size.
- NP is the class of all computable sets for which a Turing machine exists that, while not required to have such a polynomial runtime, there will exist some other Turing Machine called a "verifier."
- The verifier takes in a second input in addition to the regular input, such that for all strings x in the set, there exists a string s where $|s| = poly(|x|)$ such that s being in the second input will cause the Turing machine to accept in polynomial time. We therefore say that NP is the class of problems that have polynomial verification for "yes" answers.

Complexity Basics

- This "special string" is often called a "witness" or "certificate" and can be thought of as a "proof" for membership of the regular input, specifically a polynomial sized proof.
- It may be strange to connect proofs to computability, however all methods for doing proofs that we know of are in fact via computable inference rules - rules that can be programmed into a Turing Machine. Therefore provability "under the hood" is really just knowledge that's computable.
- P is trivially in NP - can simply choose corresponding polynomial time Turing Machine for a set in P and have any string accepted as the second input for the verifier.
- The other direction however, if $NP \subseteq P$ is unknown and known to be one of the biggest open problems in Math and Computer Science.

Complexity Basics

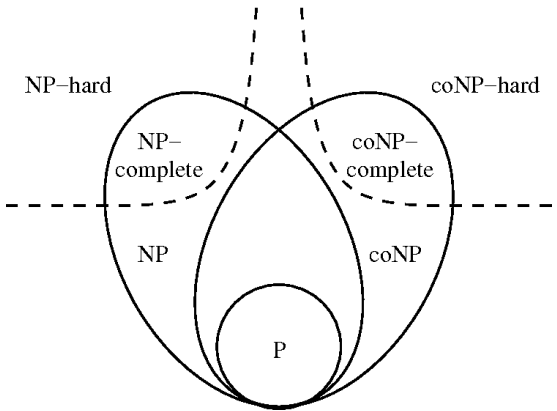
- In general the class *DTIME(T(n))* is the set of all problems that can be solved by a Turing machine with time $T(n)$ relative to input size, and likewise *DSPACE(T(n))* is the same but for the corresponding space requirement instead of the time requirement.
- For any complexity class A, the class co-A is the set of complements of decision problems in A. Namely, this means co-NP is identified by stipulating polynomial time verification for "no" answers rather than "yes" answers. It is also an open problem whether NP = co-NP.
- Intuitively most of the canonical NP problems you can think of, Boolean/Circuit satisfiability, Travelling Salesman, etc are problems with which we don't know how to do better than an exponential brute force search looking for a certain solution (or lack thereof), but which you could immediately confirm/"verify" if you actually had that solution.

Complexity Basics - Reductions

- In the context of computability theory, we say a problem Q reduces to K, or $Q \leq K$ if solving K enables us to solve Q. This is called a reduction.
- In the context of complexity theory, we say the same only if it enables us to solve Q in polynomial time. While we still call it a reduction, it's a *polynomial time reduction*, because the exact reduction/algorithm that uses our answer to K to solve Q has to run in polynomial time.
- For a class A, we say a problem is A-hard if every problem in A reduces to it. We say it's A-complete if it's A-hard and also in A.

Complexity Basics

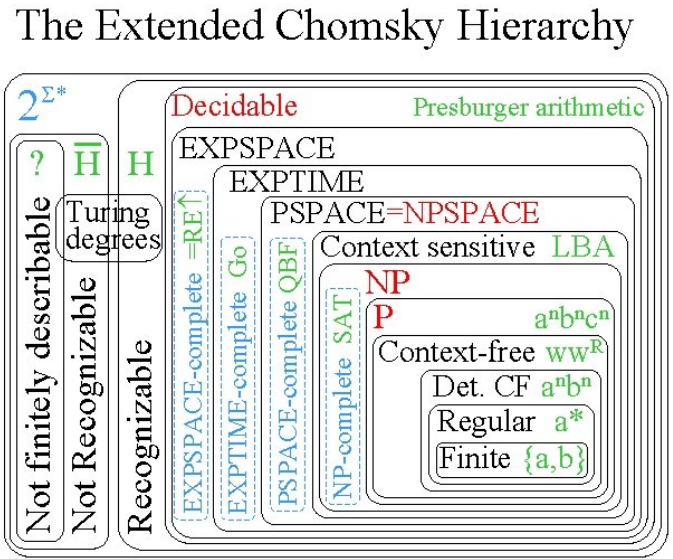
- What we think the world looks like (but have yet to prove):



Complexity Basics - Oracles

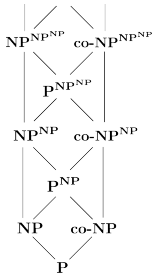
- We will often also consider computational power with *oracles*. An oracle O for a problem K is a black box we can give our machine access to, such that whenever it gives it an input it instantly outputs the answer to K for that input.
- The class A^B is the set of problems that would be in A if we allow Turing machines access to oracles for problems in B.
- Going back to computability theory, you can actually define an oracle for undecidable problems like the halting problem, and even then there are still other undecidable problems.
- Since we can keep giving oracles for those, this yields an infinite tower of classes of undecidable problems that quantifies how undecidable they are - called the arithmetic hierarchy.

Hierarchy of Complexity



Complexity Basics - Polynomial Hierarchy

- Similar to the Arithmetic Hierarchy, we can apply the same logic of building an infinite tower by successively adding oracles for the previous levels - but this time specifically to the class NP.
- What we obtain is called the polynomial hierarchy, however this has a lot more uncertainty regarding it, namely if any of the levels are distinct at all (for instance if P = NP or any two successive levels are the same, the hierarchy above it collapses).



Section 2

Quantum Computing

Qubits

- A qubit is essentially a unit vector, $\alpha|0\rangle + \beta|1\rangle$, where $\{|0\rangle, |1\rangle\}$ is our standard orthonormal basis, representing the classical bits 0 and 1 respectively, and $\alpha, \beta \in \mathbb{C}$.
- Whereas a qubit may be equal to one of those bits, in general it's neither, and we refer to it as a *superposition* of those states, until we apply what's called *measurement*.
- Once we do that, it will be a 0 with probability α^2 and a 1 with probability β^2.
- Note that the stipulation that it's a unit vector implies $\alpha^2 + \beta^2 = 1$.

Quantum Gates

- We can further perform logical operations on qubits like we could for classical bits, these "quantum gates" are essentially unitary matrices.
- A matrix being unitary means it has an inverse, and namely it's inverse is it's conjugate transpose.
- It furthermore has the property that it preserves inner products and norms, so our vector will remain a unit vector.

Multiple Qubits

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle$$
$$|\Phi\rangle = \gamma|0\rangle + \delta|1\rangle$$

- If we have multiple qubits together like above, we can define it as one state via the tensor product:

$$|\Psi\rangle|\Phi\rangle = |\Psi\rangle \otimes |\Phi\rangle = |\Psi\Phi\rangle$$

$$|\Psi\Phi\rangle = \big(\alpha|0\rangle + \beta|1\rangle\big) \otimes \big(\gamma|0\rangle + \delta|1\rangle\big)$$
$$\alpha\gamma|00\rangle + \alpha\delta|01\rangle + \beta\gamma|10\rangle + \beta\delta|11\rangle$$

- Note that the sum of the squares of the coefficients is still equal to 1.

Multiple Qubits Continued

- In general the state vector of n qubits is expressed as $\sum_{x \in \{0,1\}^n} \alpha_x |x\rangle$ where $\sum_{x \in \{0,1\}^n} \alpha_x^2 = 1$.
- While we can apply unitaries/gate operations to a multi qubit system, note that for n qubits our overall vector is in $\mathbb{C}^{2^n}$, and so our unitary matrix will be $2^n \times 2^n$ dimensions.

Quantum Circuits

- While a quantum turing machine is a well defined object, because of simplicity we often prefer to work in what's called the quantum circuit model.
- We say that a problem is computable by classical circuits if there exists a *uniform* circuit family $\{C_n\}_{n\in\mathbb{N}}$ such for any n bit input, C_n outputs a 1 for "yes" answers and a 0 for "no" answers.
- The uniformity requirement means there exists a Turing machine that on input 1^n outputs the description of C_n. This is important since arbitrary circuit families $\{C_n\}_{n\in\mathbb{N}}$ could implement any boolean function, thus solving undecidable problems which we don't want.

Quantum Circuits Continued

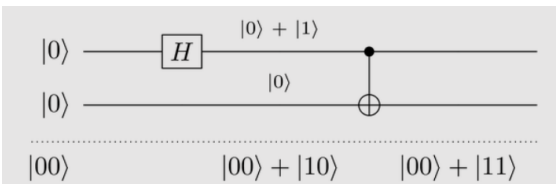
- Quantum circuit families are going to be similarly defined, but on qubits instead of bits (and such that they're initially not in superposition, so the inputs can be considered the same bitstring inputs as classical circuits) and using unitaries instead of logic gates.
- Unlike classical circuits however where we can have gates that take in multiple bits and return just one, this is not possible with quantum gates, especially since unitaries are completely reversible.
- Therefore quantum computation will always output the same number of bits as it takes.
- Normally in addition to our input bitstring, there will be extra ancilliary qubits all initialized to $|0\rangle$ that serve as extra "work" qubits in addition to one qubit designated as the output qubit that gives our output when measured at the end.

Quantum Circuits Continued

- Note that the required extra work qubits needed will always be linear to the input bitstring's length.
- It's important to still have one designated output qubit since we're still dealing with decision problems, problems with which we care about a boolean "yes" or "no" answer to.
- Famous gates: Hadamard, or H gate takes $|0\rangle$ to $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle$ and $|1\rangle$ to $\frac{1}{\sqrt{2}}|0\rangle - \frac{1}{\sqrt{2}}|1\rangle$, producing an equal superposition between the two bits.
- The CNOT gate applies to two qubits, and flips the second qubit if and only if the first qubit is $|1\rangle$, while the toffoli, or CCNOT gate is a 3 qubit gate that flips the third qubit if and only if the first 2 qubits are both 1.

Entanglement

- Two qubits are *entangled* if they cannot be represented as the tensor product of any other 2 qubits. Essentially it's making them strongly correlated, and in such a way that the whole is essentially inseparable from the part.
- The most famous way to do this is using a Hadamard followed by a CNOT gate, as in the below example (note that the coefficients are ommitted but are always $\frac{1}{\sqrt{2}}$ for the superposition states below).



BQP and Simulating Classical Circuits

- Note that due to Quantum Mechanics being inherently probabilistic, we often define acceptence with respect to a certain probability. Specifically, the class BQP, or bounded error quantum polynomial time is the class of problems that can be solved correctly with greater than $\frac{2}{3}$ probability on quantum circuits where the size of the circuit (number of gates) is $poly(n)$.
- Note that by simply repeating and taking majority vote a polynomial number of times, the acceptance probability can be brought arbitrarily close to 1. This is called *amplification*
- It can further be proved that for any problem that can be solved using classical circuits, there exists quantum circuits that also solve it with $\geq \frac{2}{3}$ probability using a polynomial number of H and $CCNOT$ gates relative to size of the classical circuit.

Why care about Quantum?

- The main reason to care about Quantum computing is because it represents the most serious challenge we have yet to the *Extended Church Turing Thesis*: the idea that the Turing machine not only can do anything that's physically "computable," but can do it within polynomial time of any other physically possible model of computation.
- Most famously, Shor's algorithm is a quantum polynomial algorithm for finding a prime factorization, a well known NP problem.
- This hints at something, either there is a very real quantum advantage that exists to the fundamental Turing machine model, which will especially be the case if we can find a quantum algorithm for NP complete problems.

Why care about Quantum?

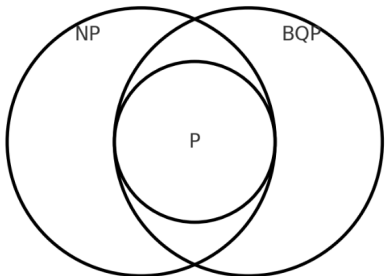
- Alternatively maybe factoring does have a classical polynomial algorithm, (though I personally think that's doubtful given how much time has failed to find one).
- It could also be the case that it has quantum advantage but only for solving certain NP "intermediate" problems that factoring happens to be one of. If that's the case, then it would be helpful to study quantum computing, if not for anything then to understand the structure of what makes these problems different.

Section 3

Oracle Separation of NP and BQP

NP and BQP

- Though they both contain P, it is unknown whether $NP \subseteq BQP$ or $BQP \subseteq NP$.
- It is believed however that they are incomparable with respect to set inclusion - that is they each have their own respective exclusive problems.
- Therefore their relationship is believed to look like this:



Simon's Problem

- Often the cases where quantum advantage are most exemplified are in "promise problems," problems where we're given some black box with a guaranteed promise about it's properties.
- Simon's problem is this: given a function (and black box for it) $f : \{0,1\}^n \to \{0,1\}^n$ that's guaranteed to exclusively match one of two properties:
  - Either injection, $f(x) = f(y) \Rightarrow x = y$
  - Or not injective but a two-to-one Simon's function, meaning $\exists s \neq 0$ such that $\forall x, y, f(x) = f(y) \Leftrightarrow y = x \oplus s$

  The problem is to decide which one it is.

Oracle Separations

- Simon's theorem: $\exists$ a quantum algorithm solving this problem with $O(n)$ queries (to the given black box that computes f), while any classical algorithm would require $\Theta(2^{\frac{n}{2}})$ queries (the quantum algorithm later actually served as inspiration for Shor's algorithm).
- While there is no proof that $BQP \nsubseteq NP$, we can use Simon's problem to prove there exists an oracle O such that $BQP^O \nsubseteq NP^O$.
- This is important since these *oracle separations* at the very least offer heuristic evidence for the sets being different. We will proceed to show this proof.

BQP^O not in NP^O

- Let $\{f_n\}_{n\in\mathbb{N}}$ be a sequence of functions $f_n : \{0,1\}^n \to \{0,1\}^n$ such that

$$f_n = \begin{cases} \text{uniform } 1-1 & \text{with probability } 0.5 \\ \text{uniform simon's function} & \text{with probability } 0.5 \end{cases}$$

- Our oracle $O : \{0,1\}^* \to \{0,1\}^*$ is such that on input x such that $|x| = k, O(x) = f_k(x)$.
- We will consider the language $coSimon^O = \{1^n | f_n \text{ is a one to one function}\}$

BQP^O not in NP^O Continued

- Since Simon's theorem establishes an $O(n)$ quantum algorithm that decides Simon's problem - that is solves it for both "yes" and "no" cases with probability greater than $\frac{2}{3}$, $coSimon^O$ is clearly in BQP^O.
- The reason we're considering this instead of just Simon's problem is because the secret string s can be used as a certificate for an NP machine to solve Simon's problem/efficiently verify any "yes" answer to a function being a Simon's function.
- On input 1^n simply choose a length n input x, compute $y = x \oplus s$ which guarantees that $x \neq y$ since s is not zero, and then you'd simply verify that $O(x) = O(y)$ (which implies $f_n(x) = f_n(y)$).
- It is not clear there is such a certificate for the "no" answers - the cases where f_n is one to one, so we might hope that this can be exploited to show $coSimon^O \notin NP^O$.

BQP^O not in NP^O Continued

- The key insight is we're going to actually build the oracle O and sequence $\{f_n\}_{n\in\mathbb{N}}$ that $coSimon^O$ depends on in it's definition, such that O and $\{f_n\}_{n\in\mathbb{N}}$ still meet the criteria we've specified their definitions must meet.
- However they will be constructed in such a way that $coSimon^O$ is not empty and no possible deterministic Turing machine M with oracle access to O can verify "yes" answers to $coSimon^O$ in polynomial time - which would prove it is not in NP^O.
- This will essentially be done via a diagonal argument.

BQP^O not in NP^O Continued

Theorem

$\exists O$ such that $BQP^O \nsubseteq NP^O$

Proof

- Let $M_1, M_2, ...$ be an enumeration of all Turing machines that can query some oracle (note that should we choose to replace an oracle on the same machine, it's algorithm will be the exact same - there is no difference other than the oracle's outputs and any behavior contingent on that).
- Let $p_i(n)$ be the time M_i takes which is some $poly(n)$
- Starting with $i = 1$, for each machine M_i, we choose the smallest number n_i such that f_{n_i} has not yet been defined and such that $p_i(n_i) \leq \sqrt{2^{n_i - 1}}$ (as that's smaller than the lower bound on number of queries needed to classically guarantee an answer for what f_{n_i} is).

BQP^O not in NP^O Continued

Proof

- We choose an injective function to assign to f_{n_i}. For values of n in between all the n_i's we choose, we also choose their respective f_n functions arbitrarily, however unlike the f_{n_i} functions those get left alone for the rest of our construction.
- Now we run M_i on 1^{n_i}. If the output is a "no" answer for f_{n_i} being a one to one function, that is clearly incorrect and we do nothing.
- If it's a "yes" answer, we reassign f_{n_i} to be a Simon's function, but one specifically constructed so that the output of all the values in $\{0,1\}^{n_i}$ which were queried by M_i are left unchanged (remember that this is acceptable with the function still being different, since the number of possible queries is far less than total possible inputs by being less than or equal to $\sqrt{2^{n_i-1}} < 2^{n_i}$).

BQP^O not in NP^O Continued

Proof

- This implies the machine will have the exact same behavior with this function, meaning on some NP certificate a "yes" will be output even though it should now be a "no" since it's a Simon's function.
- Therefore we have guaranteed that this oracle O and respective sequence we've constructed $\{f_n\}_{n\in\mathbb{N}}$ will be such that it is not possible for any M_i to accept $coSimon^O$ with polynomial time verification for "yes" answers. For each one there will be some input 1^{n_i} that causes it to fail.
- So $BQP^O \nsubseteq NP^O$. QED

Final Remarks

- There have been many oracle separations over the years in Quantum Complexity Theory, including a similar one to show $NP^O \not\subseteq BQP^O$.
- Most significantly, in recent years it's even been found that there is an oracle separation showing $BQP^O \not\subseteq PH^O$.
- The implications of even these heuristic arguments for what could be actual separations are endless, and I hope like me you see how it's an exciting time for the field!

The End

- "Turning to quantum mechanics...secret, secret, close the doors! we always have had a great deal of difficulty in understanding the world view that quantum mechanics represents... It has not yet become obvious to me that there's no real problem. I cannot define the real problem, therefore I suspect there's no real problem, but I'm not sure there's no real problem. So that's why I like to investigate things." - Richard Feynman
- Classes to take if you love this stuff: CS 474, CS 475, CS 579, special topics CS quantum computing and quantum complexity offerings.