Quantum computing and computational complexity

Gerardo Pelosi and Alessandro Barenghi

Dipartimento di Elettronica, Informazione e Bioingegneria - DEIB
Politecnico di Milano

March 16, 2022

Quantum computing advantages

Quantum computing in a nutshell

- A computing model where it is possible to act on a superimposition of exponentially many objects
- Measure only one of them after the computation

Goals of today (and next time)

1. What is the set of problems where we actually improve efficiency?
    - And in which set is it actually worth doing so?
2. What is the maximum known speedup w.r.t. classic computing?
3. What is the least guaranteed speedup w.r.t. classic computing?
    - and for which set of problems?
4. What is the link between classical randomized algorithms and quantum algorithms?

Roadmap

1. Understand a quantitative framework to define classical complexity
2. Understand classical complexity of nondeterministic computation
   - Will allow us to define boundaries for quantum computation
3. Understand classical complexity of probabilistic computation
   - Will allow us to define "the best classical mimicries"
4. Understand quantum computation complexity

Our analytic tool: complexity theory

- Complexity theory is used to classify problems according to how expensive in time/space is solving them
- It allows to
  - Provide upper/lower bounds on the time taken to solve a given problem
  - Provide upper/lower bounds on the space taken to solve a given problem
  - Understand which problems can be mapped onto equivalents (reductions)
  - Prove substantial differences between set of problems (separations)

No possibility to perform better than $O(n \log_2(n))$ for ordering problems

There's an order relation of difficulties among problems $\Rightarrow$ Order implies also equivalence $\Rightarrow$ We can make spartition among problems.

What is a problem? (for computational complexity theory)

the description of a solution is just an encoded version of the input.
the solution itself is the decryption of the description.

Definitions

1. A *problem* is a relation with both range and image $\subseteq \mathbb{N}$ (domain = codomain = $\mathbb{N}$)
   - Analogously, range and image over sets of binary strings $\subseteq \{0,1\}^*$
2. *Total and computable* relation $\rightarrow$ a finitely described algorithm written before seeing the input, computes its output (=image element) in finite time for any input(=range element) for a relation where domain and range coincide
3. Usually, we consider monodrome relations: $\exists! y \in \mathbb{N}$ s.t. $x \mapsto y$,

↳ We consider total relations ← same domain and Range

Taxonomy

- Search problem: given an input x to a problem $f(x)$, find $y = f(x), y \in \mathbb{N}$.
  - E.g.: compute the square of x, compute $x.x$ (x concatenated to itself)
- Decision problem: given x decide if x abides to some property $f(x) : \mathbb{N} \rightarrow \{0,1\}$.
  - E.g.: Is x a perfect square? Is x a palindrome?

Complexity classes

Computation model

- A k-tapes ($k > 0$) Turing Machine or a RAM machine is used as a computation model
  - The speedup thm states that changing tape encoding or number of tapes yields at most a multiplicative constant change in complexity
  - Moving from TM to RAM does not "change significantly" results
  - A bridge to the circuit complexity is a TM printing at runtime fixed input circuit solving the problem for the given input
- We evaluate the computation complexity as a function of the input length n
  - If the input is a binary encoded integer x, the complexity will be a function of $\lceil \log_2(x+1) \rceil$
  - Sometimes a unary encoding of the input integer x is used. In that case, the complexity will be a function of x
  - If the input is a Boolean string s the complexity is a function of $|s|$

Complexity classes

Temporal complexity definitions

- DTIME $(f(n))$: the set of problems for which a Deterministic TM takes $f(n)$ transitions (a.k.a, steps, moves) to compute the solution
- NTIME $(f(n))$: the set of problems for which a Non-Deterministic TM takes $f(n)$ moves to compute the solution of a problem in NTIME $(f(n))$

  A form of parallel computation.

  - Recall: a ND-TM is a TM which, if multiple transitions are possible, it computes all of them simultaneously. Its configuration is represented by a set of D-TM configurations of the active computations
- DTIME $(f(n)) \subseteq$ NTIME $(f(n))$, but it is not always known if the inclusion is proper for a given $f(n)$ (notable exception DTIME $(\mathcal{O}(n)) \subsetneq$ NTIME $(\mathcal{O}(n))$)

  Each problem solved by a DTM in linear time, can be computed by an NTM in same time.
  ↳ Not true for other complexities.
  ↳ We have no clue for that.
- Trivially, NTIME $(f(n)) \subseteq$ DTIME $\left(c^{f(n)}\right)$ for a constant $c > 1$

  ↳ This can be achieved through computing all the possible paths.

Spatial complexity definitions

- The definitions of $\texttt{DSPACE}(f(n))$, $\texttt{DSPACE}(f(n))$ are analogous to the ones of $\texttt{DTIME}(f(n))$ and $\texttt{NTIME}(f(n))$
    - replace the number of transitions with the number of tape slots written
    - for a k tapes TM, it is customary to pick the sum of the space consumption among the k memory tapes
- The input value size is conventionally not counted in the space consumption
    - It would be impossible to have sub-linear space complexity
    - Notable exception: single tape TMs ($\neq k = 1$ tapes TMs)
- Notable result 1: $\texttt{DTIME}(f(n)) \subsetneq \texttt{DSPACE}(f(n))$ in [7]
- Notable result 2: $\texttt{DSPACE}(f(n)) \subsetneq \texttt{DSPACE}(f(n)\log(f(n)))$ in [9]

Deterministic complexity classes

Notable deterministic time complexity classes

- $\mathrm{P} = \bigcup_{i\geq 1} \mathtt{DTIME}\left(n^i\right), i \in \mathbb{N}$: "practically treatable" for any n
- $\mathtt{PSPACE} = \bigcup_{i\geq 1} \mathtt{DSPACE}\left(n^i\right), i \in \mathbb{N}$: usually **not** practically tractable
  - Find the optimal chess strategy (the number of moves to finish a game is polynomial)
- $\mathtt{EXP} = \bigcup_{i\geq 1} \mathtt{DTIME}\left(2^{n^i}\right), i \in \mathbb{N}$ a.k.a. $\mathrm{EXPTIME}$ never practically treatable for large n

Time hierarchy theorem [6]

- Statement: $\mathtt{DTIME}\left(o\left(\frac{f(n)}{\log(f(n))}\right)\right) \subsetneq \mathtt{DTIME}(f(n))$
- Given $f(n)$, we can always build a problem $\notin \mathtt{DTIME}(f(n))$
- $\mathrm{P} \subsetneq \mathtt{EXP}$

Nondeterministic Polynomial time (NP) class

Definition

It is possible to define the NP class in two ways

1. $\mathtt{NP} = \bigcup_{i\geq 1} \mathtt{NTIME}\left(n^i\right), i \in \mathbb{N}$
2. Problems for which a deterministic TM verifies that a solution is ok, in polynomial time
   - ex. for decision problems: there is a deterministic TM which, given an element making the ND-TM accept, tests that it is actually one of the elements which should make the ND-TM accept

Example

State if a labyrinth (maze) has an exit, without a top-down map:

1. A non-deterministic TM will find the poly-length exiting path, if any, taking all the branches in parallel
2. A deterministic TM, given the path, will verify that it actually exits from the labyrinth through walking through it

Notable inclusions and open problems

$$\mathtt{P} \subseteq \mathtt{NP} \subseteq \mathtt{PSPACE} \subsetneq \mathtt{EXP} \quad \text{and, we know,} \quad \mathtt{P} \subsetneq \mathtt{EXP}$$

- $\mathtt{P} \subseteq \mathtt{NP}$ is simple: the ND-TM emulates a D-TM without any overhead
- $\mathtt{NP} \subseteq \mathtt{PSPACE}$: A backtracking algorithm, given arbitrary time and polyspace emulates any ND-TM computing an $\mathtt{NP}$ problem in poly space
- $\mathtt{PSPACE} \subsetneq \mathtt{EXP}$: A TM can easily enumerate all configurations taken by one solving a $\mathtt{PSPACE}$ problem in exponential time. The converse is not possible.
- The open question: $\mathtt{P} \stackrel{?}{=} \mathtt{NP}$ Likely answer: No.
    - In case $\mathtt{P} = \mathtt{NP}$, would it give $\mathtt{P} \stackrel{?}{=} \mathtt{PSPACE}$: Likely no.

Computational reductions

A tool for problem comparison

- Given two computable problems, A and B, we say that A reduces to B if, taking for granted a solver M_B for B we can build a solver M_A for A
- Operatively, we assume that M_A invokes M_B as a subprocedure
- We can state:
  - A cannot be "harder to solve" than B
  - B can be "harder to solve" than A
  - Therefore, $A \leq^T B$ (where T reminds it's a Turing reduction)

Complexity preserving (poly time, log space) reductions, a.k.a. P Cook reductions

- To have reductions "preserve" the computational complexity (denoted as $\leq_p^T$):
  - M_A makes a poly number of calls to M_B
  - M_A uses at most $\mathcal{O}(\log(n))$ extra space
- A P Cook reduction making a single call to M_B is known as a Karp reduction

Computational reductions - 2

Polynomial equivalence

- Consider two problems A and B for which both $A \leq_p^T B$ and $B \leq_p^T A$ hold
- A and B are said to be polynomially (computationally) equivalent
  - If a solver for A is found, it is "practical" to solve B too and vice versa
  - Direct consequence: if either problem is $\in \mathrm{P}$, so is the other

Examples

- Multiplying two integers is polynomially equivalent to adding them
- The inner product of vectors is polynomially equivalent to multiplying matrices

Computational classes as equivalence classes

CLASS-hardness

- Given a complexity class CLASS and a generic problem B, B is CLASS-hard iff:
  - For any problem $A \in$ CLASS we have that A reduces to B, i.e., $A \leq_p^T B$
  - Solving a CLASS-hard problem solves with extra poly effort all the problems in CLASS

CLASS-completeness

- Given a complexity class CLASS and a problem B, B is CLASS-complete iff:
  - Both $B \in$ CLASS and B is CLASS-hard
- CLASS-complete problems are the "computationally hardest" ones within a class
- The nature of a CLASS-complete problem "fully describes" the other problems in the class

CLASS-complete problem examples

P-completeness examples

- The canonical P complete problem is: compute the evaluation of a Boolean circuit (n) inputs

NP-completeness examples

- SAT: Given a generic n variables Boolean formula, state if there is an assignment making the formula true [4]
- Many natural problems are $\in$ NPC: graph colorability, existence of an Hamiltonian cycle [8]

PSPACE-completeness examples

- Quantified SAT: Given a n variables Boolean formula with universal and existential quantifiers, is it true?

A taxonomy in $\mathtt{NP}$ – Assuming $\mathtt{P} \neq \mathtt{NP}$

NP-Hard problems

Any problem such that I can poly reduce to it any problem in $\mathtt{NP}$. Note, there may be search problems in this set: e.g., Traveling salesperson, generic Integer Linear Programming

$\mathtt{NP}$-intermediate: $\mathtt{NP} \setminus (\mathtt{NPC} \cup \mathtt{P})$

- This class is non empty if $\mathtt{P} \neq \mathtt{NP}$, has infinite hierarchy inside
- Some of its problems have sub-exponential solutions
- Conjectured members:
  - (decision-)Factoring $\in \mathtt{DTIME}\left(\mathcal{O}(e^{(c+o(1))n^{\frac{1}{3}}\log(n)^{\frac{2}{3}}})\right)$, (decision)-discrete logarithms
  - Graph isomorphism (proven in 2015, disproved in '17, reproven in '17 later on, $\mathcal{O}(2^{(\log(n)^3)})$)

Complement classes $\texttt{coCLASS}$

Definition

- Consider a language L and the associated decision problem A, belonging to the class $\texttt{CLASS}$. The decision problem associated to $L^C = \{0,1\}^* \setminus L$ belongs to $\texttt{coCLASS}$.

Examples

- Observe that $\texttt{P} = \texttt{coP}$: exploit deterministic acceptance criterion
  - Given a D-TM recognizer for L, it will accept or reject in polytime. Run it and answer the reverse to get the recognizer for L^C
  - Corollary: $\texttt{P} \subseteq (\texttt{NP} \cap \texttt{coNP})$
- $\texttt{NP} = \texttt{coNP}$? Open question; highly unlikely: $\texttt{NP} \neq \texttt{coNP} \rightarrow \texttt{P} \neq \texttt{NP}$.
  - The "reversal trick" does not work any longer: a ND-TM rejects only when all computation paths lead to rejection, accepts when one accepts

Avoiding a common mistake

A $(\mathtt{NP} \cap \mathtt{coNP})$ sample

- Consider A, the decision-version of factoring: given n, k, does n have a divisor x in $1 < x \leq k$ (i.e., belong to the set of integers $\mathbf{S}$ having a divisor...). It is in $(\mathtt{NP} \cap \mathtt{coNP})$:
  - A polytime D-TM can check if a given x is a divisor of n (tests a solution for $n \in \mathbf{S}$)
  - A polytime D-TM can check if n is prime (tests a solution for $n \in (\mathbb{N} \setminus \mathbf{S})$)

A very different case

- Consider the set of graphs $L = L_1 \cap L_2$ with: set of L_1 graphs with a tour $\leq b$, set of L_1 graphs with a tour $> b$
  - Solving the problem $I \overset{?}{\in} L_1$ is in $\mathtt{NP}$ (easy to verify that a given solution tour is $\leq b$)
  - Solving the problem $I \overset{?}{\in} L_2$ is in $\mathtt{coNP}$ (from the above and the definition of $\mathtt{coNP}$)
- Solving the problem $I \overset{?}{\in} L$ is not in $(\mathtt{NP} \cap \mathtt{coNP})$!

Oracle classes

TMs with Oracles

- Either D-TMs or N-TMs with a special tape, the *query* tape
- When an input is written on the query tape, the machine invokes an oracle which solves a problem considering the contents of the query tape as input *in a single transition*
- The cost of running the oracle is constant (in time) and in space (some models take into account the query length, we won't)

Notes

- Given a solver A in class $\mathtt{C}$, and an oracle O in class $\mathtt{C}'$ the complexity class of the solver with the oracle is denoted as $\mathtt{C}^{\mathtt{O}}$
- Useful to consider $\mathtt{CLASS}$-complete oracles: we can switch to $\mathtt{C}^{\mathtt{C}'}$ for the notation
- Given that the computation of the oracle is "free", we have $\mathtt{C}^{\mathtt{C}'} = \mathtt{C}^{\mathtt{coC}'}$

The Polynomial Hierarchy (PH)

- Define a hierarchy with the relations:
  - $\Delta_{i+1} = \mathtt{P}^{\Sigma_i}$
  - $\Sigma_{i+1} = \mathtt{NP}^{\Sigma_i}$
  - $\Pi_{i+1} = \mathtt{NP}^{\Sigma_i}$
- We know $\mathtt{PH} \subseteq \mathtt{PSPACE}$
  - If $\mathtt{PH} = \mathtt{PSPACE}$, then $\mathtt{PH}$ collapses to a finite height
- If there is a $\mathtt{PH}$-complete problem $\mathtt{PH}$ collapses to a finite height
- Strong conjecture: $\mathtt{PH}$ does not collapse
- Sample Δ_2-complete problem: given a Boolean formula, does its last (in lex-order) satisfying assignment end in 1?

$$\begin{tikzcd}
 & \vdots & \\
\mathtt{NP}^{\mathtt{NP}} = \Sigma_2 & & \Pi_2 = \mathtt{coNP}^{\mathtt{NP}} \\
 & \mathtt{P}^{\mathtt{NP}} = \Delta_2 \arrow[ul] \arrow[ur] & \\
\mathtt{NP}^{\mathrm{P}} = \mathtt{NP} = \Sigma_1 \arrow[uu] \arrow[ur] & & \Pi_1 = \mathtt{coNP}^{\mathrm{P}} = \mathtt{coNP} \arrow[uu] \arrow[ul] \\
 & \Delta_0\mathrm{P} = \Sigma_0\mathrm{P} = \Pi_0\mathrm{P} = \mathrm{P} = \mathrm{P}^{\mathrm{P}} = \Delta_1\mathrm{P} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Classical Probabilistic Complexity

Nondeterministic computation $\neq$ Probabilistic computation

- ND machines perform all computations in parallel, probabilistic machines perform one of the possible computations, with a given probability
- QCs are different from both models; we will gain clarity in comparisons

Randomized algorithms

- Simply a TM having access to a tape filled with *truly random* bits (a.k.a., coins)
- What may get randomized (with a distribution over the value of the random bits)?
  - running time: Algorithm A runs in randomized time on a given input e.g., with a certain probability it terminates in poly(n) time
  - correctness: Algorithm A runs in fixed time, but returns the correct answer with probability p

↳ From now on we assume computation runs in fixed time, and we focus on the correctness of the result.

Probabilistic Polynomial time (PP) class [5]

Definition

PP class: (*probabilistic poly time*) the problem is solved in $\mathtt{DTIME}(poly(n))$ by an algorithm outputting the correct answer with $\Pr > \frac{1}{2}$ $P(c = head) = p$ $(H, T) \rightarrow H$

$(T, H) \rightarrow T$

Observations

- First attempt to define problems tractable with randomized computation
- Class way wider than expected. Case in point, it contains $\mathtt{NP}$-complete!
- You can solve SAT in polytime with $\Pr > \frac{1}{2}$ as follows. Given a Boolean formula:
  - Return "yes" with $\Pr > \frac{1}{2} - \frac{1}{2^{2n}}$
  - Otherwise, draw a random value for the variables, test if it satisfies the formula, and answer "yes" if it does, "no" if it does not
  - You provide the correct answer with $\Pr > \frac{1}{2} - \frac{1}{2^{2n}} + \frac{1}{2^n} > \frac{1}{2}$

Bounded error Probabilistic Polynomial time (BPP) class

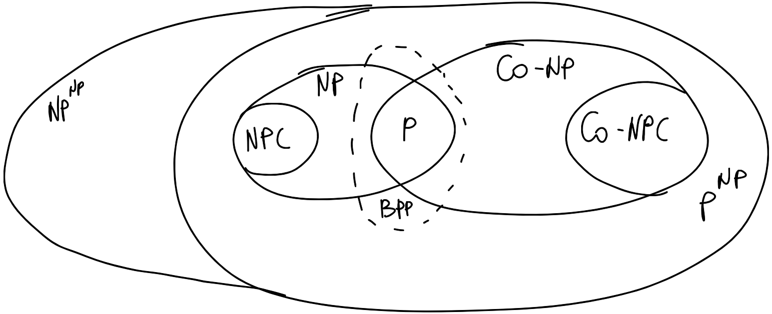
Definition

- BPP class: the problem is solved in $\mathtt{DTIME}(poly(n))$ by an algorithm outputting the correct answer with $\Pr > \frac{1}{2} + k$, where $\frac{1}{2} + \frac{1}{poly(n)} < k < 1 - \frac{1}{2^n}$

$(> \frac{2}{3})$

Observations

- The choice of the lower bound for k is such that calling the solver $poly(n)$ times I can decide the correct answer through majority voting
- The choice of the upper bound for k is to make the BPP solver distinguishable from a deterministic one with a $poly(n)$ number of calls
- We will pick $k = \frac{1}{4}$ or $k = \frac{1}{6}$ to get a "pleasant" Pr (either $\frac{3}{4}$ or $\frac{2}{3}$) of a correct answer



NP^NP
NP
NPC
P
BPP
Co-NP
Co-NPC
P^NP

Relations with non probabilistic classes

$\mathrm{P} \subseteq \mathrm{BPP}$

- Yes: solve the problem and then produce a wrong answer on purpose with low probability
- Mounting evidence that $\mathrm{P} = \mathrm{BPP}$ may hold (swap random coins with *pseudorandom* coins)

$\mathrm{BPP} \subseteq \mathrm{PP}$

Yes: the constraint on the correctness of the solution provided by a BPP solver is tighter than the one of a PP solver

$\mathrm{BPP} \subseteq \mathrm{NP}$?

- Open question: is randomness "a substitute for parallelism"? Likely, no (see $\mathrm{P} \stackrel{?}{=} \mathrm{BPP}$)
- However, we know that $\mathrm{BPP} \subseteq \mathrm{NP}^{\mathrm{NP}} = \Sigma_2$ [3]

A bigger picture

Relations with PH

- BPP is surely contained in PH, as it is contained in Σ_2
- We do not know if PP is contained in PH per se but:
  - There is an oracle O s.t. $\Delta_2^O \not\subseteq \mathrm{PP}^O$ [1]
  - A fundamental result by Toda showed that $\mathrm{PH} \subseteq \mathrm{P}^{\mathrm{PP}}$
- Our insight improves considering $\sharp\mathrm{P}$: the class of functions which compute *how many* accepting paths a ND-TM has
  - A $\sharp\mathrm{P}$-complete problem is $\sharp SAT$, i.e., counting how many satisfying assignments does a Boolean formula have
- It is known that $\mathrm{P}^{\mathrm{PP}} = \mathrm{P}^{\sharp\mathrm{P}}$ (colloquially, a counting oracle does the same work of a probabilistic polynomial oracle to det. polynomial solver)
  - As a consequence, we get the inclusion $\mathrm{PH} \subseteq \mathrm{P}^{\sharp\mathrm{P}}$

Randomness and nondeterminism - 2

- We tackled the question of whether randomized computation can somehow make up for the parallelism of nondeterministic communication (TL;DR, no, likely $\mathtt{BPP} \not\subseteq \mathtt{NP}$)
- What about the converse, $\mathtt{NP} \overset{?}{\subseteq} \mathtt{BPP}$?
  - Colloquially: is there a way to substitute classical randomness with the parallelism provided by nondeterminism?
- Contrary to some intuition, the answer is likely to be no
  - Understanding the strongest argument we have requires us to consider another computation model: computing with advices

Nonuniform classes and advices

An ideal algorithm for each length

- Up to now, we restricted our solvers to employ the same algorithm for all the inputs
- Is it possible to improve the complexity if we change this?
  - Trivially yes: have one algorithm for each input *value*: everything becomes constant-time as the algorithm just outputs the solution
  - As the former is too much, we consider the case of one algorithm *per input length*

Non-uniform complexity

- A way to model the "one-algorithm-per-input-length" behaviour is to consider a solver receiving a fixed $f(n)$ long *advice* string together with each n bit input
- The complexity of a class-$\mathtt{C}$ solver with an $f(n)$ long advice is denoted as $\mathtt{C}/f(n)$
  - It will trivially contain $\mathtt{C}$: the solver just ignores the advice
- The advices are assumed to be generated together with the solver

A single bit of advice goes a long way...

P class
with 1 bit of advice.

Example

- Consider the $\underbrace{\texttt{P}/1}$ non uniform class. How far does a single bit of advice go?
- Claim: the following problem has a solver in $\texttt{P}/1$:
  - Given an input string of the form 1^n (n repetitions of the character 1), compute if the n-th TM halts when run with input n.
- Proof: Just consider the advice bit to be the answer!
- The above problem is way outside $\texttt{PH}$ and $\texttt{PP}$: it's a unary-encoded variant of the halting problem! It's not even computable without advice!

- Is granting advice too much? Does $\texttt{P}/1$ include all problems?

...but not all the way

- Denote as $\mathtt{P}/poly$ the nonuniform class where the advice can be as long as any polynomial function of the input length n.
- We show that there is a (computable, Boolean) function outside $\mathtt{P}/poly$

$\mathtt{P}/n^{\log(n)} \neq ALL$

- Consider a list of poly time TMs taking an $n^{\log n}$ bit advice out of the set $\mathbf{S}$ of $n^{\log n}$ bit strings, to compute a function on an n bit input.
- Claim: there is a Boolean function of n inputs that the set $\mathbf{M}$ of the first n machines in the list fail to compute.
- Proof: consider Boolean functions of n inputs, there are 2^{2^n} of them
- The functions computed by the first n TMs at hand are only $|\mathbf{S}||\mathbf{M}| = n2^{n^{\log(n)}} < 2^{2^n}$

Advices, nondeterminism and randomness

Advices and nondeterminism

- Conjecture: $NP \not\subseteq P/poly$: poly advice is not enough to match nondeterminism
  - Note that $\mathtt{P} = \mathtt{NP}$ would imply the opposite
  - However, even assuming $\mathtt{P} \neq \mathtt{NP}$ we have not proven $NP \not\subseteq P/poly$ yet
  - Karp and Lipton proved $NP \subseteq P/poly$ implies $\mathtt{PH} = \Sigma_2 = \mathtt{NP}^{\mathtt{NP}}$ ($\mathtt{PH}$ collapses to 2nd level)
  - We are quite convinced that poly advice is no match for nondeterminism

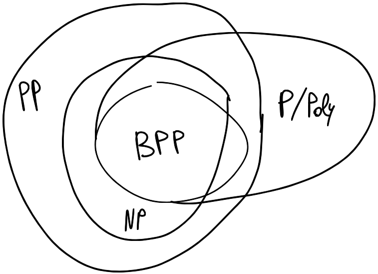
Advices and randomness

- $BPP \subseteq P/poly$: proven by Adleman
  - Consider a $\mathtt{BPP}$ solver, run it n^2 times, the error prob. over all coin outcomes is $\approx \frac{1}{2^{n^2}}$
  - Note that there are at most 2^n inputs, for the solver, and for each one, at most $\frac{1}{2^{n^2}}$ random coin strings cause an error
  - By Union Bound, at most a fraction of $\frac{2^n}{2^{n^2}} < 1$ of the coin outcomes ever cause an error
  - Pick one of the never-error-causing coin strings and feed it as an advice to a $\mathtt{P}/poly$ solver!

Putting the pieces together

Back to $\mathtt{NP} \subseteq \mathtt{BPP}$?

- Assume that $\mathtt{P} \neq \mathtt{NP}$ (or the answer is trivially yes)
- Since $BPP \subseteq P/poly$, $\mathtt{NP} \subseteq \mathtt{BPP}$ would imply $\mathtt{NP} \subseteq P/poly$
- Thus $\mathtt{NP} \subseteq \mathtt{BPP}$ (by Karp-Lipton) implies $\mathtt{PH} = \Sigma_2$ (which we do not think possible)
- It is very unlikely that randomized computation is as powerful as nondeterminism
  - Understanding if the "natural randomness" of quantum measurements does better or worse one of our goals.


PP
P/Poly
BPP
NP

Moving onto Quantum complexity theory

Choosing a quantum computation model

- To evaluate the complexity of quantum algorithms we have to choose our cost function and computation model.
- To make at least some sense, our model should take into account the fact that quantum computations are intrinsically probabilistic
  - The very result of the measurements is defined in probability
- It is useful to keep in mind that our probabilistic measurement gives us access to a bounded amount of the information contained in a quantum state (Holevo Bound)

Moving onto Quantum complexity theory

Choosing a quantum computation model

1. Quantum Turing Machine model [2]
   - The "usual" Turing machine; each tape slot can contain either a classic symbol or a qubit
   - Cost in number of transitions, used tape slots on the memory tapes as usual
2. Quantum circuit gate [10]
   - A quantum circuit made of quantum gates, solving the problem for any input
   - Cost criterion: time in number of gates/circuit depth, space in number of qubits
     - The Solovay-Kitaev theorem guarantees us that the gate set choice is not asymp. relevant
3. Quantum query model
   - Consider a quantum circuit computing a function f of your choice
     - The circuit can be computed on any superimposition of your choice
   - Time cost as the number of whole circuit computations (queries)
     - Useful when the size of the circuit does not depend on the input size

Probabilistic computational classes (for quantum computers)

BQP: Bounded error Quantum Polynomial

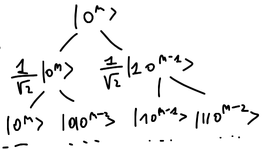
Each problem in BQP is solved in DTIME(poly(N))

$BQP \subseteq PSPACE$

Consider simulating a quantum computer with a classical one.

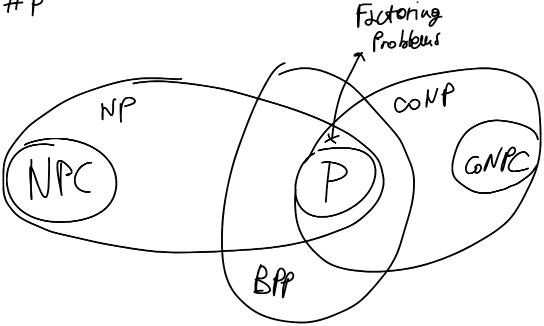
Consider $|0^n\rangle \leftarrow$ polynomial space

What if we apply H gate to only the first Q-bit? $\Rightarrow$



The idea is to split computation in branches from the root of a three $\Rightarrow$ Then iterating a depth first search and backtracking.

$BQP \subseteq P^{\#P}$



$coNP = NP$

$\Downarrow$

$NPC \cap coNP = \emptyset$

Textbook references

- Sanjeev Arora and Boaz Barak - *Computational Complexity: A Modern Approach*, Cambridge University Press
- Free draft available at `https://theory.cs.princeton.edu/complexity/book.pdf`

Bibliography I

Richard Beigel.
Perceptrons, pp, and the polynomial hierarchy.
Comput. Complex., 4:339–349, 1994.

Ethan Bernstein and Umesh V. Vazirani.
Quantum complexity theory.
SIAM J. Comput., 26(5):1411–1473, 1997.

Ran Canetti.
More on BPP and the polynomial-time hierarchy.
Inf. Process. Lett., 57(5):237–241, 1996.

Stephen A. Cook.
The complexity of theorem-proving procedures.
In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

John Gill.
Computational complexity of probabilistic turing machines.
SIAM J. Comput., 6(4):675–695, 1977.

Juris Hartmanis and Richard E Stearns.
On the computational complexity of algorithms.
Transactions of the American Mathematical Society, 117:285–306, 1965.

Bibliography II

John E. Hopcroft, Wolfgang J. Paul, and Leslie G. Valiant.
On time versus space.
J. ACM, 24(2):332–337, 1977.

Richard M. Karp.
Reducibility among combinatorial problems.
In Raymond E. Miller and James W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations, held March 20-22, 1972, at the IBM Thomas J. Watson Research Center, Yorktown Heights, New York, USA*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

Richard Edwin Stearns, Juris Hartmanis, and Philip M. Lewis II.
Hierarchies of memory limited computations.
In *6th Annual Symposium on Switching Circuit Theory and Logical Design, Ann Arbor, Michigan, USA, October 6-8, 1965*, pages 179–190. IEEE Computer Society, 1965.

Andrew Chi-Chih Yao.
Quantum circuit complexity.
In *34th Annual Symposium on Foundations of Computer Science, Palo Alto, California, USA, 3-5 November 1993*, pages 352–361. IEEE Computer Society, 1993.