Into the Quantum Realm: On the Collapsible Polynomial Hierarchy for Promise Problems

Advanced Algorithm Course

Abolfazl Ghorbani
Asha Soroushpoor

University of Tehran

2024/02/03

Table of Contents

The Polynomial-time Hierarchy

The Polynomial-time Hierarchy

Basic definitions and properties

Definition: Σ_k^P

$\Sigma_0^P := P$

For $k \geq 1$:

$L \in \Sigma_k^P \iff$ there are polynomials $p_1, p_2, ..., p_k$ and a polynomial-time predicate F s.t.:

$$x \in L \leftrightarrow \underset{y_1 \in S_1}{\exists} \underset{y_2 \in S_2}{\forall} \exists ... \underset{y_k \in S_k}{Q} F(x, y_1, ..., y_k)$$

where $S_i := \{0, 1\}^{p_i(|x|)}$

Definition: Π_k^P

$\Pi_0^P := P$

For $k \geq 1$:

$L \in \Pi_k^P \iff$ there are polynomials $p_1, p_2, ..., p_k$ and a polynomial-time predicate F s.t.:

$$x \in L \leftrightarrow \underset{y_1 \in S_1}{\forall} \underset{y_2 \in S_2}{\exists} \forall ... \underset{y_k \in S_k}{Q} F(x, y_1, ..., y_k)$$

where $S_i := \{0, 1\}^{p_i(|x|)}$

Definition: The Polynomial-time Hierarchy PH

$$PH := \bigcup_{i \in \mathbb{N}} \Sigma_i^P \cup \Pi_i^P$$

for all $i \in \mathbb{N}$:

1. $\Pi_i^P = co\Sigma_i^P$
2. $\Sigma_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$
3. $\Pi_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$
4. $(?)\Sigma_i^P \subset \Sigma_{i+1}^P$

for all $i \in \mathbb{N}$:

1. $\Pi_i^P = co\Sigma_i^P$
2. $\Sigma_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$
3. $\Pi_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$
4. $(?)\Sigma_i^P \subset \Sigma_{i+1}^P$

for all $i \in \mathbb{N}$:

1. $\Pi_i^P = co\Sigma_i^P$
2. $\Sigma_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$
3. $\Pi_i^P \subseteq \Sigma_{i+1}^P \cap \Pi_{i+1}^P$
4. $(?)\Sigma_i^P \subset \Sigma_{i+1}^P$

Scheme of the Polynomial-time hierarchy

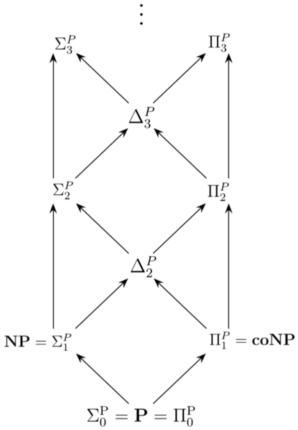


Figure: The Polynomial-time Hierarchy.

lemma

For all k, $L \in \Sigma_{k+1}^{P} \iff$ there is a polynomial p and $A \in \Pi_k^P$:

$$x \in L \leftrightarrow \underset{y \in \{0,1\}^{p(|x|)}}{\exists} < x, y >\in A$$

Proof($\Leftarrow$)

$$< x, y >\in A \leftrightarrow \underset{y_1 \in S_1}{\forall} \underset{y_2 \in S_2}{\exists} \forall \ldots \underset{y_k \in S_k}{Q} F(< x, y >, y_1, \ldots, y_k)$$

for some polynomial F and $S_i := \{0,1\}^{p(|x|)_i}$ for polynomials $p_1, p_2, \ldots, p_k$. Putting this in the equation gives us the desired result in one direction.

Equivalent definition

lemma

For all k, $L \in \Sigma_{k+1}^P \iff$ there is a polynomial p and $A \in \Pi_k^P$:

$$x \in L \leftrightarrow \underset{y \in \{0,1\}^{p(|x|)}}{\exists} < x, y > \in A$$

Proof($\Leftarrow$)

$$< x, y > \in A \leftrightarrow \underset{y_1 \in S_1}{\forall} \; \underset{y_2 \in S_2}{\exists} \; \forall \ldots \underset{y_k \in S_k}{Q} \; F(< x, y >, y_1, \ldots, y_k)$$

for some polynomial F and $S_i := \{0, 1\}^{p(|x|)_i}$ for polynomials $p_1, p_2, \ldots, p_k$. Putting this in the equation gives us the desired result in one direction.

Proof(⇒)

$$x \in L \leftrightarrow \underset{y \in S}{\exists} \ \underset{y_1 \in S_1}{\forall} \ \exists \ldots \underset{y_k \in S_k}{Q} F(x, y, y_1, \ldots, y_k)$$

Where, $S := \{0,1\}^{p(|x|)}$ and S_i as before for $i = 1, \ldots, k$.
Let

$$\hat{p}(n) := n^{deg(p)+1} + M$$

for a large enough M such that $\forall n \ \hat{p}(n) \geq p(n)$.

Proof($\Rightarrow$)

Now define:

$\hat{F}(z, y_1, \dots y_k)$

$L := min\{t \mid \hat{p}(t) + t = |z|\}$

if $L = None$ **then**

    return False

end if

return $F(z[1, .., L], z[L+1, ..., L+p(L)], y_1, \dots y_k)$

Then we have this nice property:

$$x \in L \leftrightarrow \underset{y \in S}{\exists} \ \underset{y_1 \in S_1}{\forall} \ \exists \dots \underset{y_k \in S_k}{Q} F(x, y, y_1, \dots, y_k)$$

$$\leftrightarrow \underset{\hat{y} \in \{0,1\}^{\hat{p}(x)}}{\exists} \ \underset{y_1 \in S_1}{\forall} \ \exists \dots \underset{y_k \in S_k}{Q} \hat{F}(<x, \hat{y}>, y_1, \dots, y_k)$$

Proof(⇒)

Now put:

$$A := \{z | \forall_{y_1 \in S_1} \exists \ldots Q_{y_k \in S_k} \hat{F}(z, y_1, \ldots, y_k)\}$$

But $\hat{F}$ is clearly polynomial-time, which means $A \in \Pi_k^P$. Finally:

$$x \in L \leftrightarrow \exists_{\hat{y} \in \{0,1\}^{\hat{p}(x)}} \forall_{y_1 \in S_1} \exists \ldots Q_{y_k \in S_k} \hat{F}(< x, \hat{y} >, y_1, \ldots, y_k)$$

$$\leftrightarrow \exists_{y \in \{0,1\}^{\hat{p}(|x|)}} < x, y > \in A$$

this completes the proof.

lemma

For all k, $L \in \Pi^P_{k+1} \iff$ there is a polynomial p and $A \in \Sigma^P_k$:

$$x \in L \leftrightarrow \underset{y \in \{0,1\}^{p(|x|)}}{\forall} < x, y > \in A$$

Theorem: PH is collapsible

$i \geq 1$ & $\Pi^P_i = \Sigma^P_i \rightarrow PH = \Sigma^P_i$

lemma

For all $k, L \in \Pi^P_{k+1} \iff$ there is a polynomial p and $A \in \Sigma^P_k$:

$$x \in L \leftrightarrow \underset{y \in \{0,1\}^{p(|x|)}}{\forall} < x, y > \in A$$

Theorem: PH is collapsible

$i \geq 1 \ \& \ \Pi^P_i = \Sigma^P_i \rightarrow PH = \Sigma^P_i$

Proof.

It suffices to show that $i \geq 1$ & $\Sigma_i^P = \Pi_i^P \to \Sigma_i^P = \Sigma_{i+1}^P$:
$L \in \Sigma_{i+1}^P \iff$ there is a polynomial p and $A \in \Pi_i^P$:

$$x \in L \leftrightarrow \underset{|y|=p(|x|)}{\exists} < x, y > \in A$$

$\iff$ there are $A \in \Sigma_i^P$ and p: ...
$\iff$ there are $p, p^{'}$ and $B \in \Pi_{i-1}^P$:

$$x \in L \leftrightarrow \underset{|y|=p(|x|)}{\exists} \underset{|y^{'}|=p^{'}(|x|)}{\exists} < x, y, y^{'} > \in B$$

$\iff$ there are $p^{''} = p + p^{'}$ and $B \in \Pi_{i-1}^P$:

$$x \in L \leftrightarrow \underset{|y^{''}|=p"(|x|)}{\exists} < x, y^{''} > \in B$$

$\iff L \in \Sigma_i^P$. □

The Polynomial-time Hierarchy

Relationships to other complexity classes

Review: NP

$L \in NP \iff$ there is a polynomial p and a polynomial-time predicate F s.t.:

$$x \in L \iff \underset{|y| \leq p(|x|)}{\exists} F(x, y)$$

Lemma

$NP = \Sigma_1^P$

Proof

1. for one direction use padding to fix the length of y
2. for the other direction check the length of y in algorithm

Review: NP

$L \in NP \iff$ there is a polynomial p and a polynomial-time predicate F s.t.:

$$x \in L \iff \underset{|y| \le p(|x|)}{\exists} F(x, y)$$

Lemma

$NP = \Sigma_1^P$

Proof

1. for one direction use padding to fix the length of y
2. for the other direction check the length of y in algorithm

Review: NP

$L \in NP \iff$ there is a polynomial p and a polynomial-time predicate F s.t.:

$$x \in L \iff \exists_{|y| \leq p(|x|)} F(x, y)$$

Lemma

$NP = \Sigma_1^P$

Proof

1. for one direction use padding to fix the length of y
2. for the other direction check the length of y in algorithm

Definition: BPP

$L \in BPP \iff$ there is a polynomial p and a polynomial-time randomized algorithm F s.t.:

$$x \in L \implies \Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r)) \geq 2/3$$

$$x \notin L \implies \Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r)) \leq 1/3$$

	$F(x,r)$	$\neg F(x,r)$
$x \in L$	$\geq 2/3$	$\leq 1/3$
$x \notin L$	$\leq 1/3$	$\geq 2/3$

Table: $\Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r))$

lemma

BPP is closed under complement.

Proof.

Immediate from definition.

lemma

If $L \in BPP$ then there is an algorithm F such that:

$$\forall x \Pr_{r}(F(x,r) = right\ answer) \geq 1 - \frac{1}{3|x|}$$

Proof.

Since $L \in BPP$ then there is a polynomial-time two-sided Monte Carlo algorithm A, which gives the right answer with probability at least $\frac{1}{2} + \epsilon$ for some $0 < \epsilon \leq \frac{1}{2}$. Now recall the algorithm A_t from the book, which repeats A for t times and output a solution if A outputs that solution for at least $\lceil \frac{t}{2} \rceil$ times. As it is shown in the book, it is enough to set $t \geq \frac{2ln\frac{2}{3|x|}}{ln(1-4\epsilon^2)}$ to have that

$$\Pr_r\left(A_t(x, r) = right\ answer\right) \geq 1 - \frac{1}{3|x|}$$

Proof.

This means that instead of a fixed t in A_t, it's enough that the algorithm calculates the proper t as above, and then continues. Clearly this calculation takes place in polynomial- time. On the other hand it's enough to have $t \in O(|ln\frac{2}{3|x|}|) = O(ln(|x|))$, and since A runs in polynomial-time, then repeating A for $O(ln(|x|))$ is again a polynomial-time task. This completes the proof.

Theorem

$BPP \subseteq \Sigma_2$

Proof.

Let $L \in BPP$ and F has the property in the previous lemma and let $m = |x|$. We show the following:

$$x \in L \iff \exists y_1, y_2, ..., y_m \in \{0,1\}^m \forall z \in \{0,1\}^m \bigvee_{i=1}^{m} F(x, y_i \oplus z)$$

Theorem

$BPP \subseteq \Sigma_2$

Proof.

Let $L \in BPP$ and F has the property in the previous lemma and let $m = |x|$. We show the following:

$$x \in L \iff \exists y_1, y_2, \ldots, y_m \in \{0,1\}^m \forall z \in \{0,1\}^m \bigvee_{i=1}^{m} F(x, y_i \oplus z)$$

Proof.

$\Rightarrow$: Suppose $x \in L$, then

$$\Pr_{y_1,y_2,\dots,y_m}\left(\forall z \in \{0,1\}^m \bigvee_{i=1}^{m} F(x, y_i \oplus z)\right)$$

$$= 1 - \Pr_{y_1,y_2,\dots,y_m}\left(\exists z \in \{0,1\}^m \bigwedge_{i=1}^{m} \neg F(x, y_i \oplus z)\right)$$

$$\geq 1 - \sum_{z\in\{0,1\}^m} \Pr_{y_1,y_2,\dots,y_m}\left(\bigwedge_{i=1}^{m} \neg F(x, y_i \oplus z)\right)$$

$$\geq 1 - \frac{2^m}{(3m)^m} > 0.$$

This implies that there are $y_1, y_2, \dots, y_m$ such that:

$$\forall z \in \{0,1\}^m \bigvee_{i=1}^{m} F(x, y_i \oplus z)$$

Proof.

$\Leftarrow$: Suppose $x \notin L$, then for an arbitrary sequence $y_1, ... y_m \in \{0, 1\}^m$

$$
\begin{aligned}
\Pr_z \left(\bigwedge_{i=1}^{m} \neg F(x, y_i \oplus z) \right) &= 1 - \Pr_z \left(\bigvee_{i=1}^{m} F(x, y_i \oplus z) \right) \\
&\geq 1 - \sum_{i=1}^{m} \Pr_z \left(F(x, y_i \oplus z) \right) \\
&\geq 1 - \frac{m}{3m} > 0.
\end{aligned}
$$

Thus:

$$\forall y_1, ..., y_m \exists z \in \{0, 1\}^m \neg \bigvee_{i=1}^{m} F(x, y_i \oplus z))$$

$\square$

Sipser–Lautemann theorem

$BPP \subseteq \Sigma_2^P \cap \Pi_2^P$

Proof.

Use the previous lemma and the fact that *BPP* is closed under complement.

Theorem

$PH \subseteq PSPACE$

Proof

It is sufficient to show that for any i if $\Pi_i \subseteq PSPACE$ then $\Sigma_{i+1} \subseteq PSPACE$:
Suppose $L \in \Sigma_{i+1}$ then there is a polynomial p and $A \in \Pi_i^P$:

$$x \in L \leftrightarrow \exists_{y \in \{0,1\}^{p(|x|)}} < x, y > \in A$$

A is decidable in $PSPACE$ by hypothesis, thus $\exists_{y \in \{0,1\}^{p(|x|)}} < x, y > \in A$ is decidable in $PSPACE$ using the odometer method.

Lemma

For all i: Σ_i^P is closed under $\leq_p$

Proof

$A \leq_p B \iff$ there is a polynomial algorithm R:

$$x \in A \iff R(x) \in B$$

Now suppose $B \in \Sigma_i^P$ then:

$$x \in A \leftrightarrow \underset{y_1 \in S_1}{\exists} \; \underset{y_2 \in S_2}{\forall} \; \exists \ldots \underset{y_k \in S_k}{Q} F(R(x), y_1, \ldots, y_k)$$

where $S_i := \{0, 1\}^{p_i(|R(x)|)}$

Relationships to other complexity classes

Lemma

For all i: Σ_i^P is closed under $\leq_p$

Proof

$A \leq_p B \iff$ there is a polynomial algorithm R:

$$x \in A \iff R(x) \in B$$

Now suppose $B \in \Sigma_i^P$ then:

$$x \in A \leftrightarrow \underset{y_1 \in S_1}{\exists} \underset{y_2 \in S_2}{\forall} \exists \ldots \underset{y_k \in S_k}{Q} F(R(x), y_1, \ldots, y_k)$$

where $S_i := \{0, 1\}^{p_i(|R(x)|)}$

Theorem

$PH = PSPACE \implies PH$ Collapses.

Proof

$PH = PSPACE \implies PH$ have a complete problem which is in Σ_i for some i and since Σ_i is closed under $\leq_p$: $\Sigma_i = PH$

Theorem

$PH = PSPACE \implies PH$ Collapses.

Proof

$PH = PSPACE \implies PH$ have a complete problem which is in Σ_i for some i and since Σ_i is closed under $\leq_p$: $\Sigma_i = PH$

1. $NP = \Sigma_1^P$ & $CoNP = \Pi_1^P$
2. $P = NP \iff P = PH$
3. $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$
4. $PH \subseteq PSPACE$
5. $PH = PSPACE \implies PH$ Collapses.

1. $NP = \Sigma_1^P$ & $CoNP = \Pi_1^P$
2. $P = NP \iff P = PH$
3. $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$
4. $PH \subseteq PSPACE$
5. $PH = PSPACE \implies PH$ Collapses.

1. $NP = \Sigma_1^P$ & $CoNP = \Pi_1^P$
2. $P = NP \iff P = PH$
3. $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$
4. $PH \subseteq PSPACE$
5. $PH = PSPACE \implies PH$ Collapses.

1. $NP = \Sigma_1^P$ & $CoNP = \Pi_1^P$
2. $P = NP \iff P = PH$
3. $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$
4. $PH \subseteq PSPACE$
5. $PH = PSPACE \implies PH$ Collapses.

1. $NP = \Sigma_1^P$ & $CoNP = \Pi_1^P$
2. $P = NP \iff P = PH$
3. $BPP \subseteq \Sigma_2^P \cap \Pi_2^P$
4. $PH \subseteq PSPACE$
5. $PH = PSPACE \implies PH$ Collapses.

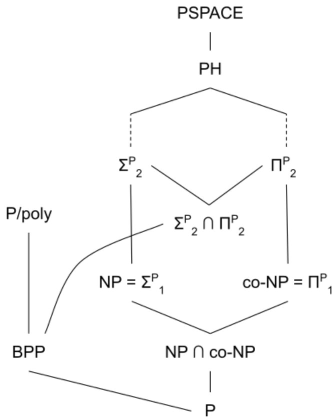


Figure: Complexity Classes Inclusion Tree

Promise Problems

Promise Problems

Introduction

”How many of the readers have learned about promise problems in an undergraduate ’theory of computation’ course or even in a graduate course on complexity theory?”

”Scant few? And yet I contend that almost all readers refer to this notion when thinking about computational problems, although they may be often unaware of this fact.

”How many of the readers have learned about promise problems in an undergraduate ’theory of computation’ course or even in a graduate course on complexity theory?”
”Scant few? And yet I contend that almost all readers refer to this notion when thinking about computational problems, although they may be often unaware of this fact.

review

A language L over some alphabet Σ is simply a subset of Σ^*. In the world of computer science, problems are often formalized with the corresponding language L over $\{0, 1\}^*$ using some interpretation (e.g. any graph can be converted to some string in $\{0, 1\}^*$ uniquely). But are such interpretations bijective (e.g. does any string in $\{0, 1\}^*$ represent a graph)?

question

What should the decider Turing machine do if the given string doesn't represent any instance of the problem?

Example

Consider any standard entry like "given a planar graph, determine whether or not ...". A more formal statement will refer to strings that represent planar graphs. Either way, one may wonder what should the decision procedure do when the input is not a (string representing a) planar graph.

question

What should the decider Turing machine do if the given string doesn't represent any instance of the problem?

Example

Consider any standard entry like "given a planar graph, determine whether or not ...". A more formal statement will refer to strings that represent planar graphs. Either way, one may wonder what should the decision procedure do when the input is not a (string representing a) planar graph.

first approach

One common formalistic answer is that all strings are interpreted as representations of planar graphs (typically, by using a decoding convention by which every "non-canonical" representation is interpreted as a representation of some fixed planar graph).

Second approach

Another (even more) formalistic "solution" is to discuss the problem of distinguishing yes-instances from anything else (i.e., effectively viewing strings that violate the promise as no-instances).

first approach

One common formalistic answer is that all strings are interpreted as representations of planar graphs (typically, by using a decoding convention by which every “non-canonical” representation is interpreted as a representation of some fixed planar graph).

Second approach

Another (even more) formalistic “solution” is to discuss the problem of distinguishing yes-instances from anything else (i.e., effectively viewing strings that violate the promise as no-instances).

Downsides of these conventions

Both conventions miss the true nature of the original computational problem, which is concerned with distinguishing planar graphs of one type from planar graphs of another type (i.e., the complementary type). Moreover, the complexity of the problem can be drastically affected [1]. Consider a computational problem that, analogously to the one above, reads "Given a Hamiltonian graph, determine whether or not ..." Deciding whether a graph is Hamiltonian is NP-complete and doesn't seem to be tractable.

[1] Maybe this is one of the reasons that we often focus on polynomial-time procedures.

Definition: promise problems

A promise problem A is a pair of sets, denoted (A_+, A_-), s.t.:
$A_+, A_- \subseteq \{0,1\}^*$ and $A_+ \cap A_- = \emptyset$.
The set $A_+ \cup A_-$ is called the promise.

Definition: Solving a promise problem

A Turing machine T solve the problem A if:

- $x \in A_+ \implies T \textit{ accepts } x$
- $x \in A_- \implies T \textit{ rejects } x$

T can output anything outside of the promise, or even doesn't halt.

Definition: Solving a promise problem

A Turing machine T solve the problem A if:

- $x \in A_+ \implies T \text{ accepts } x$
- $x \in A_- \implies T \text{ rejects } x$

T can output anything outside of the promise, or even doesn't halt.

Promise problem corresponded to a language

$A_+ := L$ & $A_- := \bar{L}$

proposition

T decides $(A_+, A_-) \iff T$ decides L.

Promise problem corresponded to a language

$A_+ := L$ & $A_- := \bar{L}$

proposition

T decides $(A_+, A_-) \iff T$ decides L.

Definition: being a special case of another problem

Problem A is a special case of problem B if:

$$A_+ \subseteq B_+ \ \& \ A_- \subseteq B_-$$

proposition

T solves B and A is a special case of $B \implies T$ solves A.

Definition: being a special case of another problem

Problem A is a special case of problem B if:

$$A_+ \subseteq B_+ \;\&\; A_- \subseteq B_-$$

proposition

T solves B and A is a special case of B $\implies$ T solves A.

Promise Problems

Technical Applications

Review: BPP

$L \in BPP \iff$ there is a polynomial p and a polynomial-time randomized algorithm (predicate) F s.t.:

$$x \in L \implies \Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r)) \geq 2/3$$

$$x \notin L \implies \Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r)) \leq 1/3$$

Fact

BPP is not likely to have any complete problems.

Review: BPP

$L \in BPP \iff$ there is a polynomial p and a polynomial-time randomized algorithm (predicate) F s.t.:

$$x \in L \implies \Pr_{r \in \{0,1\}^{p(|x|)}}(F(x,r)) \geq 2/3$$

$$x \notin L \implies \Pr_{r \in \{0,1\}^{p(|x|)}}(F(x,r)) \leq 1/3$$

Fact

BPP is not likely to have any complete problems.

Definition: $Promise - BPP$

$L \in Promise - BPP \iff$ there is a polynomial p and a polynomial-time randomized algorithm (predicate) F s.t.:

$$x \in L_+ \implies \Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r)) \geq 2/3$$

$$x \in L_- \implies \Pr_{r \in \{0,1\}^{p(|x|)}} (F(x,r)) \leq 1/3$$

Fact

A complete for *Promise – BPP*:

$$(M, x, 1^p) \implies \Pi_+ = \{M \textit{ is a probabilistic machine that accepts } x \textit{ with probability at least } 2/3 \textit{ in } p \textit{ steps}\}$$

$$(M, x, 1^p) \implies \Pi_- = \{M \textit{ is a probabilistic machine that rejects } x \textit{ with probability at least } 2/3 \textit{ in } p \textit{ steps}\}$$

Alert

The definition of polynomial-time reduction should be revised for the promise version of the problems

Fact

A complete for *Promise – BPP*:

$(M, x, 1^p) \implies \Pi_+ = \{M \textit{ is a probabilistic machine that accepts } x \textit{ with probability at least } 2/3 \textit{ in } p \textit{ steps}\}$

$(M, x, 1^p) \implies \Pi_- = \{M \textit{ is a probabilistic machine that rejects } x \textit{ with probability at least } 2/3 \textit{ in } p \textit{ steps}\}$

Alert

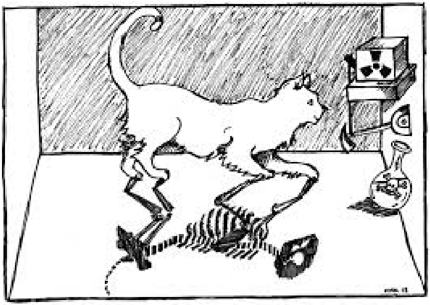
The definition of polynomial-time reduction should be revised for the promise version of the problems

Enjoy the next section...

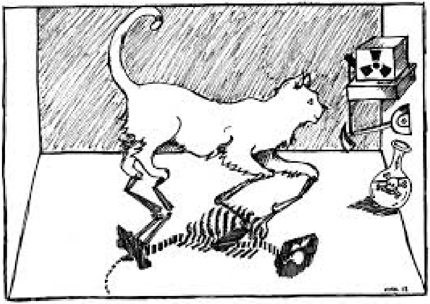
Quantum Complexity

Quantum Complexity

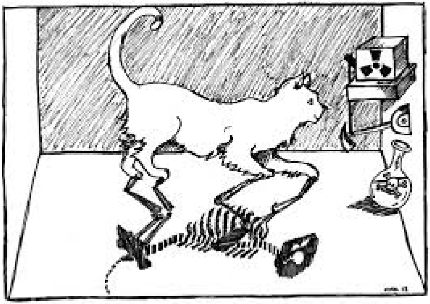
Quantum Mechanics



- Assume there is a cat in a box, with a handle that will break the poison bottle based on the reaction of a quantum particle



- Assume there is a cat in a box, with a handle that will break the poison bottle based on the reaction of a quantum particle
- Is the cat dead or alive?



- Assume there is a cat in a box, with a handle that will break the poison bottle based on the reaction of a quantum particle
- Is the cat dead or alive?
- Neither, the cat is bound to the SuperPosition of the quantum particle, being both dead and alive at the same time.

- Niels Bohr (Copenhagen interpretation):
  In essence this interpretation amounts to a sophisticated way of saying not to ask the question. Science relies on the notion of measurements and observations, so in essence asking in quantum mechanics “what is a measurement?” is equivalent to asking the axioms of euclidean geometry “what is a point?”.

- Hugh Everett (Many worlds interpretation):
  There is only one process in quantum mechanics, unitary evolution. What we perceive as measurements is just unitary evolution applied to the measuring equipment and the observer. Essentially the universe splits every time a measurement is performed, and one copy sees $|0\rangle$ while the other sees $|1\rangle$.

- David Bohm (Non-local hidden variables):
  The third answer says that both of the previous answers are unacceptable, so quantum mechanics is somehow incomplete in the sense that there is an additional aspect that we're missing. Non-local hidden variables is one proposal to fill that gap, but there are others.

Quantum Complexity

Quantum Computing

- We represent the Superposition of data with a vector in complex space with length of 1.
- We represent the value of the vector with the unit basis of our space.
- For the purpose of this lecture we follow the traditional representation by Qubits:
  - We represent one qubit with $|x\rangle$ with definition:
    $|0\rangle := \begin{bmatrix} 1 \\ 0 \end{bmatrix} \; |1\rangle := \begin{bmatrix} 0 \\ 1 \end{bmatrix}$
  - We define n not entangled qubit with this notation:
    $|x\rangle^{\oplus n} := \underbrace{|x\rangle|x\rangle \ldots |x\rangle}_{n}$

- Similarly we represent n entangled multi-qubits with n basis vectors

$$|00\rangle := \begin{bmatrix}1\\0\\0\\0\end{bmatrix} \; |01\rangle := \begin{bmatrix}0\\1\\0\\0\end{bmatrix} \; |10\rangle := \begin{bmatrix}0\\0\\1\\0\end{bmatrix} \; |11\rangle := \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$$

Let us give an example of what a quantum operation looks like. We represent quantum operations by multiplying our data by an L2-norm preserving matrix e.g unitary matrices
For example an elementary operation is controlled-NOT (CNOT) that is:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

it maps $|00\rangle \rightarrow |00\rangle$, $|01\rangle \rightarrow |01\rangle$ $|10\rangle \rightarrow |11\rangle$, $|11\rangle \rightarrow |10\rangle$. in other language it performs not on the second bit iff the first bit is true or it maps $|xy\rangle \rightarrow |x(x \oplus y)\rangle$

We can measure our quantum state by projecting our quantum state vector onto one of our *Orthonormal Basis*. The probability of successful projection is equal to Coefficient power 2. for example in our data representation the matrix

$$\begin{bmatrix} \frac{1}{\sqrt{3}} \\ \sqrt{\frac{2}{3}} \end{bmatrix} = \frac{1}{\sqrt{3}}|0\rangle + \sqrt{\frac{2}{3}}|1\rangle$$

So when we measure we get $|0\rangle$ with the probability of $\frac{1}{3}$ and we get $|1\rangle$ with probability of $\frac{2}{3}$.

Quantum state

A quantum state is an n-dimensional complex vector with the length of 1.

Quantum basis state

Quantum basis states are a collection of **Orthonormal basis** which we can measure our quantum state with them.

Measurement

Assume we have a multi-qubit like:
$|a_1 a_2 \dots a_n\rangle$ where $\sum_{n=1}^{n} a_i^2 = 1$ we can measure this qubit in basis states where the probability of us measuring this qubit onto *ith* state is a_i^2

Quantum state

A quantum state is an n-dimensional complex vector with the length of 1.

Quantum basis state

Quantum basis states are a collection of **Orthonormal basis** which we can measure our quantum state with them.

Measurement

Assume we have a multi-qubit like:
$|a_1 a_2 \dots a_n\rangle$ where $\sum_{n=1}^{n} a_i^2 = 1$ we can measure this qubit in basis states where the probability of us measuring this qubit onto *ith* state is a_i^2

Quantum state

A quantum state is an n-dimensional complex vector with the length of 1.

Quantum basis state

Quantum basis states are a collection of **Orthonormal basis** which we can measure our quantum state with them.

Measurement

Assume we have a multi-qubit like:
$|a_1 a_2 \dots a_n\rangle$ where $\sum_{n=1}^{n} a_i^2 = 1$ we can measure this qubit in basis states where the probability of us measuring this qubit onto *ith* state is a_i^2

example

Assume we can perform a quantum operation which rotates our quantum state by 45^o the matrix resembling this operation is as follows $\frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}$ performing this operation twice is equivalent to the NOT operation:

$$|x\rangle \rightarrow \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ -1 \end{bmatrix} \rightarrow |1\rangle, |1\rangle \rightarrow \frac{1}{\sqrt{2}}\begin{bmatrix} 1 \\ 1 \end{bmatrix} \rightarrow |0\rangle$$

but if we perform this operation once then measure and perform it again we get $|0\rangle$ or $|1\rangle$

Assume we have a 2-qubit :

$$\alpha|00\rangle + \beta|01\rangle + \gamma|10\rangle + \delta|11\rangle$$

we can perform an operation only on the second qubit by finding the tensor product of our transformation:

$$\frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1\\ 1 & 1\end{bmatrix} \otimes \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & -1 & 0 & 0\\ 1 & 1 & 0 & 0\\ 0 & 0 & 1 & -1\\ 0 & 0 & 1 & 1\end{bmatrix}$$

the probability of us measuring the first qubit $|0\rangle$ is equal to $\alpha^2 + \beta^2$. Assuming we measured the first one $|0\rangle$ the state of the second qubit collapses to:

$$\frac{\alpha|0\rangle + \beta|1\rangle}{\sqrt{\alpha^2 + \beta^2}}$$

Proof

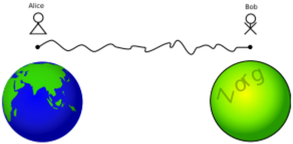
$$x_2 = \alpha' |0\rangle + \beta' |1\rangle \tag{1}$$

$$\beta'^2 = P(x_2 = |1\rangle | x_1 = |0\rangle) = \frac{P(x_2 = |1\rangle \wedge x_1 = |0\rangle)}{P(x_1 = |0\rangle)} = \frac{\beta^2}{\alpha^2 + \beta^2} \tag{2}$$

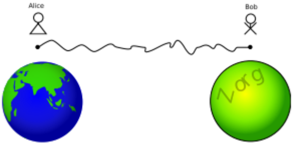
$$\alpha'^2 = P(x_2 = |0\rangle | x_1 = |0\rangle) = \frac{P(x_2 = |0\rangle \wedge x_1 = |0\rangle)}{P(x_1 = |0\rangle)} = \frac{\alpha^2}{\alpha^2 + \beta^2} \tag{3}$$

Bell's inequality

- In 1935, Einstein, Podolsky and Rosen wrote a famous paper where they brought to widespread attention the tension between quantum mechanics and relativity. One thing that relativity says is that you can't send information faster than light.
- In 1964 Bell played the roll of a quantum complexity theorist. He said, let's compare entangle ment against classical correlation as resources to perform some task.



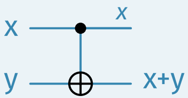
- In 1935, Einstein, Podolsky and Rosen wrote a famous paper where they brought to widespread attention the tension between quantum mechanics and relativity. One thing that relativity says is that you can't send information faster than light.
- In 1964 Bell played the roll of a quantum complexity theorist. He said, let's compare entangle ment against classical correlation as resources to perform some task.



Quantum Circuits

We present a series of quantum operations on qubits via a circuit which each operation represents itself as a gate in our circuit

CNOT gate



Hadamard Gate

Hadarmand gate switches between $|0\rangle, |1\rangle$ and $|0\rangle + |1\rangle, |0\rangle - |1\rangle$ basis. The matrix for this operation is as follows: $\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$ We represent this operation by following gate:

Tofoli's Gate (CCNOT)

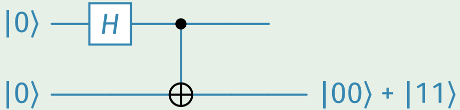
$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

$$(a, b, c) \rightarrow (a, b, c \oplus (a \wedge b))$$

Quantum Circuits

Circuit example 1

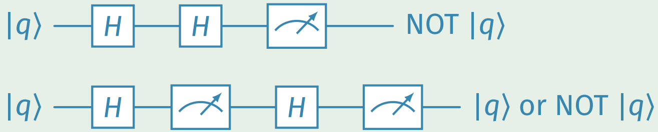
From what we learned we can introduce a circuit starting from unentangled states creating entangled states[2].



[2] https://en.wikipedia.org/wiki/Bell_state

Circuit example 2

The example from before about measurement between computations is as follows:



- What is the quantum computation model?
- This question was answered in rigorous terms by Bernstein and Vzirani in a 70 page paper in 1993 who defined a quantum Turing machine that could have the tape head and symbols and superpositions.
- Independently, Andy Yao defined quantum circuits, which are computationally equivalent. This is what people use today because they are simpler.

- What is the quantum computation model?
- This question was answered in rigorous terms by Bernstein and Vzirani in a 70 page paper in 1993 who defined a quantum Turing machine that could have the tape head and symbols and superpositions.
- Independently, Andy Yao defined quantum circuits, which are computationally equivalent. This is what people use today because they are simpler.

- What is the quantum computation model?
- This question was answered in rigorous terms by Bernstein and Vzirani in a 70 page paper in 1993 who defined a quantum Turing machine that could have the tape head and symbols and superpositions.
- Independently, Andy Yao defined quantum circuits, which are computationally equivalent. This is what people use today because they are simpler.

Quantum Complexity

Quantum Complexity

Reminder

P

P is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a Turing machine M and a polynomial q so that for inputs $x \in \{0,1\}^n$, M Terminates in at most $q(n)$ steps and accepts iff $x \in L$

PSPACE

PSPACE is defined like P except we're limited by $q(n)$ tape cells, rather than time.

EXP

EXP is defined like P, but we're limited by $2^{q(n)}$ steps.

Reminder

P

P is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a Turing machine M and a polynomial q so that for inputs $x \in \{0,1\}^n$, M Terminates in at most $q(n)$ steps and accepts iff $x \in L$

PSPACE

PSPACE is defined like P except we're limited by $q(n)$ tape cells, rather than time.

EXP

EXP is defined like P, but we're limited by $2^{q(n)}$ steps.

P

P is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a Turing machine M and a polynomial q so that for inputs $x \in \{0,1\}^n$, M Terminates in at most $q(n)$ steps and accepts iff $x \in L$

PSPACE

PSPACE is defined like P except we're limited by $q(n)$ tape cells, rather than time.

EXP

EXP is defined like P, but we're limited by $2^{q(n)}$ steps.

BPP

P is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a Turing machine **probabilistic** M and a polynomial q so that for inputs $x \in \{0,1\}^n$, M Terminates in at most $q(n)$ steps and

- if $x \in L$, then M accepts with probability > 2/3
- if $x \notin L$, then M accepts with probability < 1/3

Or equivalently there exists a Turing machine M with two inputs x,r both in $\{0,1\}^n$ which

- if $x \in L$, then M(x,r) accepts for at least 2/3 values of r.
- if $x \notin L$, then M(x,r) accept for at most 1/3 values of r.

constants importance

the constants are not important; we can amplify the success probability as much as we want by chaining Turing machines.

Reminder

BPP

P is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a Turing machine **probabilistic** M and a polynomial q so that for inputs $x \in \{0,1\}^n$, M Terminates in at most $q(n)$ steps and

- if $x \in L$, then M accepts with probability > 2/3
- if $x \notin L$, then M accepts with probability < 1/3

Or equivalently there exists a Turing machine M with two inputs x,r both in $\{0,1\}^n$ which

- if $x \in L$, then M(x,r) accepts for at least 2/3 values of r.
- if $x \notin L$, then M(x,r) accept for at most 1/3 values of r.

constants importance

the constants are not important; we can amplify the success probability as much as we want by chaining Turing machines.

Definition

BQP is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a *uniform* family of polynomial-size quantum circuit $\{C_n\}$ over some basis of universal gates and a polynomial q so that for all n and inputs $x \in \{0,1\}^n$.

- if $x \in L$, then $C_n(|x\rangle|0\rangle^{\otimes q(n)})$ accepts with probability > $2/3$
- if $x \notin L$, then $C_n(|x\rangle|0\rangle^{\otimes q(n)})$ accepts with probability < $1/3$

Difference with BPP

Unlike BPP we cannot extract the element of randomness out of our definition of a quantum circuit. This is the fundamental difference of quantum complexity classes and traditional ones.

BQP

Definition

BQP is the class of languages $L \subseteq \{0,1\}^*$ for which there exists a *uniform* family of polynomial-size quantum circuit $\{C_n\}$ over some basis of universal gates and a polynomial q so that for all n and inputs $x \in \{0,1\}^n$.

- if $x \in L$, then $C_n(|x\rangle|0\rangle^{\otimes q(n)})$ accepts with probability $> 2/3$
- if $x \notin L$, then $C_n(|x\rangle|0\rangle^{\otimes q(n)})$ accepts with probability $< 1/3$

Difference with BPP

Unlike BPP we cannot extract the element of randomness out of our definition of a quantum circuit. This is the fundamental difference of quantum complexity classes and traditional ones.

$P \subseteq BPP$

$P \subseteq BPP$

Just don't use randomness

$P \subseteq BPP$

Just don't use randomness

$BPP \subseteq PSPACE$

BQP PROPERTIES

$P \subseteq BPP$

Just don't use randomness

$BPP \subseteq PSPACE$

Run the problem for each possible r and keep count of how many are accepted.

BQP PROPERTIES

$P \subseteq BPP$

Just don't use randomness

$BPP \subseteq PSPACE$

Run the problem for each possible r and keep count of how many are accepted.

$PSPACE \subseteq EXP$

BQP properties

$P \subseteq BPP$

Just don't use randomness

$BPP \subseteq PSPACE$

Run the problem for each possible r and keep count of how many are accepted.

$PSPACE \subseteq EXP$

Polynomial space can go through exponentially many configurations without looping.

Solovay-Kitaev Theorem

With a finite set of gates, we can approximate any n-qubit unitary within L2 accuracy ϵ using $2^n(n + polylog(1/\epsilon))$ gates.

Solovay-Kitaev Theorem

With a finite set of gates, we can approximate any n-qubit unitary within L2 accuracy ϵ using $2^n(n + polylog(1/\epsilon))$ gates.

$SU(2)$

$$SU(2) := \begin{bmatrix} \alpha & -\bar{\beta} \\ \beta & \bar{\alpha} \end{bmatrix} : \alpha, \beta \in \mathbb{C}, |\alpha|^2 + |\beta|^2 = 1$$

Universal Quantum gates

A set of **universal quantum gates** is any set of gates to which any operation possible on a quantum computer can be reduced, that is, any other unitary operation can be expressed as a finite sequence of gates from the set.

Some examples of Universal Quantum Gate

- Rotational operators $R_x(\theta), R_y(\theta), R_z(\theta)$ and phase shift operator $P(\phi)$ and CNOT can be used to form several Universal gate set
- Tofoli(CCNOT) + Hadamard gate. The Toffoli gate alone forms a set of universal gates for reversible Boolean algebraic logic circuits, which encompass all classical computation.

formalized Solovay-Kitaev Theorem

Let G be a finite set of elements in SU(2) containing its own inverses (so ($g \in G$ implies $g^{-1} \in G$) and such that the group $\langle G \rangle$ they generate is dense in SU(2). Consider some $\epsilon > 0$. Then there is a constant c such that for any $U \in SU(2)$, there is a sequence S of gates from G of length $O(log^{c}(1/\epsilon))$ such that $||S - U|| \leq \epsilon$. That is, S approximates U to operator norm error.

$P \subseteq BQP$

[3] Well not exactly, see https://people.eecs.berkeley.edu/ vazirani/f04quantum/notes/lec5/lec5.pdf

[4] we use BQP-completness of APPROX-QCIRCUIT-PROB

BQP Properties Continued

$P \subseteq BQP$

Any classical circuit can be simulated by a quantum circuit. (for more information see chapter 3.1.2 of [6])

$BPP \subseteq BQP$

[3] Well not exactly, see https://people.eecs.berkeley.edu/ vazirani/f04quantum/notes/lec5/lec5.pdf

[4] we use BQP-completness of APPROX-QCIRCUIT-PROB

BQP Properties Continued

$P \subseteq BQP$

Any classical circuit can be simulated by a quantum circuit. (for more information see chapter 3.1.2 of [6])

$BPP \subseteq BQP$

We can generate a random number as much as we want, for example using Hadamard gate and $|0\rangle$ gives us a random bit. [3]

$BQP \subseteq EXP$

[3] Well not exactly, see https://people.eecs.berkeley.edu/ vazirani/f04quantum/notes/lec5/lec5.pdf

[4] we use BQP-completness of APPROX-QCIRCUIT-PROB

BQP Properties Continued

$P \subseteq BQP$

Any classical circuit can be simulated by a quantum circuit. (for more information see chapter 3.1.2 of [6])

$BPP \subseteq BQP$

We can generate a random number as much as we want, for example using Hadamard gate and $|0\rangle$ gives us a random bit. [3]

$BQP \subseteq EXP$

Since a quantum state can be written as $|\psi\rangle = \sum a_x |x\rangle$, we can simulate the whole evolution of quantum vectors with classical computers, within exponential time at most. [4]

[3] Well not exactly, see https://people.eecs.berkeley.edu/ vazirani/f04quantum/notes/lec5/lec5.pdf

[4] we use BQP-completness of APPROX-QCIRCUIT-PROB

BQP ⊆ PSPACE

by using Feynmann's path integral we can simulate the circuit in polynomial time. Schrodinger's picture:

$$P(x_m) = |\langle x_m | U_m U_{m-1} \dots U_1 | 0 \rangle|^2$$

While in Feynman's path algorithm $P(x_m)$ is calculated by summing up the contributions of $(2^n)^{m-1}$:

$$P(x_m) = |\langle x_m | U | 0 \rangle^n|^2 = |\sum_{x_1, x_2 \dots x_{m-1} \in \{0,1\}^n} \prod_{j=1}^{m} \langle x_j | U_j | x_{j-1} \rangle|^2$$

Schrodinger's take $m2^n$ time and 2^n space while Feynman's take 4^m time and $m + n$ space.

$P^P = P^{\#P}$

BQP Properties Continued

$P^P = P^{\#P}$

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

Since we can do counting in polynomial space, $P^{\#P} \subseteq PSPACE$. Also $\#P$ can follow all paths nondeterministic in Feynman's path integral

$P^P = P^{\#P}$

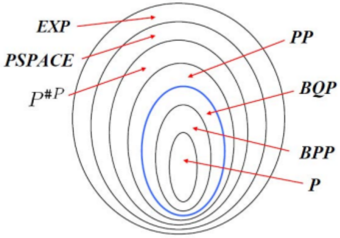
$BQP \subseteq P^{\#P} \subseteq PSPACE$

Since we can do counting in polynomial space, $P^{\#P} \subseteq PSPACE$. Also $\#P$ can follow all paths nondeterministic in Feynman's path integral

$BQP \subseteq PP$

Quantum Complexity Inclusion

$$P \subseteq BPP \subseteq BQP \subseteq PP \subseteq P^{\#P} \subseteq PSPACE \subseteq EXP$$



$BPP \neq BQP$?

Can we prove that quantum computer exceeds classical computers? The answer is no since it would imply $P \neq PSPACE$ which is as difficult to prove as $P \neq NP$

Where is NP?

We don't know!. The conjecture is that $NP \not\subseteq BQP$ which means quantum computer cannot solve NP-complete problems in polynomial time. However we have no idea whether $BQP \not\subseteq NP$ or not.

$BQP^{BQP} = BQP$

We use uncomputing.

$BPP \neq BQP$?

Can we prove that quantum computer exceeds classical computers? The answer is no since it would imply $P \neq PSPACE$ which is as difficult to prove as $P \neq NP$

Where is NP?

We don't know!. The conjecture is that $NP \not\subseteq BQP$ which means quantum computer cannot solve NP-complete problems in polynomial time. However we have no idea whether $BQP \not\subseteq NP$ or not.

$BQP^{BQP} = BQP$

We use uncomputing.

$BPP \neq BQP$?

Can we prove that quantum computer exceeds classical computers? The answer is no since it would imply $P \neq PSPACE$ which is as difficult to prove as $P \neq NP$

Where is NP?

We don't know!. The conjecture is that $NP \not\subseteq BQP$ which means quantum computer cannot solve NP-complete problems in polynomial time. However we have no idea whether $BQP \not\subseteq NP$ or not.

$BQP^{BQP} = BQP$

We use uncomputing.

Quantum Complexity

Quantum Query model

We have already discussed how proving complexity lower bounds on quantum computation seems very difficult. Just as with classical complexity, then, we turn to simplified, idealized models of computation in the hopes of proving something and gaining intuitions. We try to model, in the quantum world, the classical idea of a 'bounded-query' algorithm, one which has access to a very large (binary) input and tries to compute some predicate of the input without looking at too many input bits. We think of this input as a function $f(x) : \{0,1\}^n \rightarrow \{0,1\}$.

These query unitaries are usually restricted into two types:

controlled-not query

maps basis state $|x, w\rangle$ to $|x, w \oplus f(x)\rangle$

textitphase query

maps $|x\rangle$ to $(-1)^{f(x)}|x\rangle$

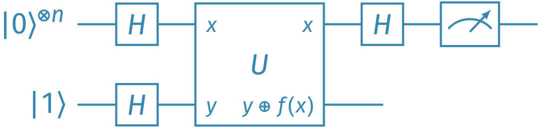
Equivalency

These two models are equivalent.

The function takes n-bit binary values as input and produces either a 0 or a 1 as output for each such value. We are **promised** that the function is either constant (0 on all inputs or 1 on all inputs) or balanced (1 for exactly half of the input domain and 0 for the other half).
The task is to find whether the oracle is constant or balanced.

Classical approach

In classical approach for n-bits input we need $2^{n-1} + 1$ queries to get the answer. We have to at least check half of the possible outcomes.



The algorithm starts with $n + 1$ bit state $|0\rangle^{\oplus n}|1\rangle$, after Hadamard's transform we obtain:

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}|x\rangle(|0\rangle - |1\rangle)$$

applying the query gives us:

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle(|0\rangle-|1\rangle)$$

let's forget about last bit so we have:

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle$$

we also know that:

$$H^{\oplus n}|k\rangle=\frac{1}{\sqrt{2^n}}\sum_{j=0}^{2^n-1}(-1)^{k.j}|j\rangle$$

where $j.k=j_0.k_0\oplus j_1.k_1\oplus\ldots j_{n-1}.k_{n-1}$ where $\oplus$ is addition module 2.

Proof

n=1:

$$|x\rangle = \begin{bmatrix}\alpha \\ \beta\end{bmatrix} = \alpha|0\rangle + \beta|1\rangle$$

$$H|x\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}\alpha + \beta \\ \alpha - \beta\end{bmatrix} = \frac{1}{\sqrt{2}}(\alpha + \beta)|0\rangle + (\alpha - \beta)|1\rangle$$

$$H|x\rangle = \frac{1}{\sqrt{2}}((-1)^{x.0}|0\rangle + (-1)^{x.1}|1\rangle)$$

$$H|x\rangle = \frac{1}{\sqrt{2}}\sum_{j=0}^{1}(-1)^{xj}|j\rangle$$

Proof

$$H^{\otimes n}|x_1, x_2 \dots x_n\rangle = \sum_{j_1, j_2 \dots j_n \in \{0,1\}} \frac{(-1)^{(x_1 j_1 + x_2 j_2 + \dots x_n j_n)} |j_1, j_2 \dots j_n\rangle}{\sqrt{2^n}}$$

Therefor applying Hadamard gate the second time gives us

$$\frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{f(x)} \left[\frac{1}{\sqrt{2^n}} \sum_{y=0}^{2^n-1} (-1)^{x.y} |y\rangle \right]$$

$$\sum_{y=0}^{2^n-1} \left[\frac{1}{2^n} \sum_{x=0}^{2^n-1} (-1)^{f(x)} (-1)^{x.y} \right] |y\rangle$$

So the probability of state $|0\rangle$ to be measured is

$$|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|^2$$

this evaluates to 1 if f is constant and 0 if f is balanced. in the other words the end measurement will be $|0\rangle^{\otimes n}$ iff $f(x)$ is constant.

Suppose that there is a Boolean function, $f : \{0,1\}^n \to \{0,1\}$. The **promise** is that, the f is in form of $s.x$ which is inner product of s and x module 2.

Classical approach

We can query basis strings and find the secret:

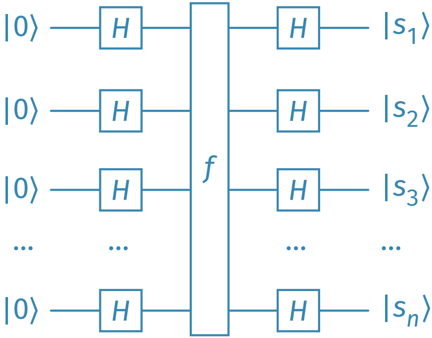
$$f(100\dots0) = s_1$$

$$f(010\dots0) = s_2$$

$$\dots$$

$$f(000\dots1) = s_n$$

Bernstein-Vazarani Algorithm



$$|0\rangle^{\otimes n} \to \frac{1}{2^n} \sum_{x\in\{0,1\}^n |x\rangle} \tag{4}$$

$$\to \frac{1}{2^n} \sum_{x\in\{0,1\}^n (-1)^{f(x)} |x\rangle} \tag{5}$$

$$= \frac{1}{2^n} \sum_{x\in\{0,1\}^n (-1)^{s.x} |x\rangle} \tag{6}$$

$$= \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{s_1 x_1} \dots (-1)^{s_n x_n} |x\rangle \tag{7}$$

Polynomial-time Hierarchy for Promise Problems

Polynomial-time Hierarchy for Promise Problems

Generalizing the polynomial-time hierarchy

Complement of a promise problems

$\bar{A} := (A_-, A_+)$

Elementwise complement of a class of promise problems

$co(C) := \{\bar{A} | A \in C\}$

Properties

1. $co(co(C)) = C$
2. $co(C) = co(C') \iff C = C'$

Complement of a promise problems

$\bar{A} := (A_-, A_+)$

Elementwise complement of a class of promise problems

$co(C) := \{\bar{A} | A \in C\}$

Properties

1. $co(co(C)) = C$
2. $co(C) = co(C') \iff C = C'$

Complement of a promise problems

$\bar{A} := (A_-, A_+)$

Elementwise complement of a class of promise problems

$co(C) := \{\bar{A} | A \in C\}$

Properties

1. $co(co(C)) = C$
2. $co(C) = co(C') \iff C = C'$

Existential quantifier

Let C be a class of promise problems. Then $L \in \exists C$ if there is a problem A in C and a polynomial p such that:

$$x \in L_+ \iff \exists_{y \in \{0,1\}^{p(|x|)}} < x, y >\in A_+$$

$$x \in L_- \iff \forall_{y \in \{0,1\}^{p(|x|)}} < x, y >\in A_-$$

Universal quantifier

Let C be a class of promise problems. Then $L \in \forall C$ if there is a problem A in C and a polynomial p such that:

$$x \in L_+ \iff \forall_{y \in \{0,1\}^{p(|x|)}} <x, y> \in A_+$$

$$x \in L_- \iff \exists_{y \in \{0,1\}^{p(|x|)}} <x, y> \in A_-$$

1. $C \subseteq \exists C$
2. $C \subseteq \forall C$
3. $co(\exists C) = \forall co(C)$
4. $\exists\exists C = \exists C$
5. $\forall\forall C = \forall C$
6. $C \subseteq C' \implies \exists C \subseteq \exists C'$
7. $C \subseteq C' \implies \forall C \subseteq \forall C'$

Properties of Quantifiers

1. $C \subseteq \exists C$
2. $C \subseteq \forall C$
3. $co(\exists C) = \forall co(C)$
4. $\exists\exists C = \exists C$
5. $\forall\forall C = \forall C$
6. $C \subseteq C' \implies \exists C \subseteq \exists C'$
7. $C \subseteq C' \implies \forall C \subseteq \forall C'$

1. $C \subseteq \exists C$
2. $C \subseteq \forall C$
3. $co(\exists C) = \forall co(C)$
4. $\exists\exists C = \exists C$
5. $\forall\forall C = \forall C$
6. $C \subseteq C' \implies \exists C \subseteq \exists C'$
7. $C \subseteq C' \implies \forall C \subseteq \forall C'$

1. $C \subseteq \exists C$
2. $C \subseteq \forall C$
3. $co(\exists C) = \forall co(C)$
4. $\exists\exists C = \exists C$
5. $\forall\forall C = \forall C$
6. $C \subseteq C' \implies \exists C \subseteq \exists C'$
7. $C \subseteq C' \implies \forall C \subseteq \forall C'$

A Collapsible Polynomial Hierarchy for promise class C

$\Sigma_0 := C$ and $\Pi_0 := C$
For $k \geq 1$:

$$\Sigma_k := \exists \Pi_{k-1} \text{ and } \Pi_k := \forall \Sigma_{k-1}$$

$H_C := \bigcup_{i \in \mathbb{N}} \Sigma_i \cup \Pi_i$

Lemma

conditional collapse $k \geq 1 \ \& \ \Sigma_k = \Pi_k \rightarrow H_C = \Sigma_k$

Proof

It suffices to show that $k \geq 1 \ \& \ \Sigma_k = \Pi_k \rightarrow \Sigma_k = \Sigma_{k+1} \ldots$

Lemma

conditional collapse $k \geq 1 \;\&\; \Sigma_k = \Pi_k \rightarrow H_C = \Sigma_k$

Proof

It suffices to show that $k \geq 1 \;\&\; \Sigma_k = \Pi_k \rightarrow \Sigma_k = \Sigma_{k+1}$...

Lemma

conditional collapse $k \geq 1$ & $\Sigma_k = \Pi_k \to H_C = \Sigma_k$

Proof

It suffices to show that $k \geq 1$ & $\Sigma_k = \Pi_k \to \Sigma_k = \Sigma_{k+1}$:
$L \in \Sigma_{k+1} \iff L \in \exists\Pi_k \iff L \in \exists\Sigma_k \iff L \in \exists\exists\Pi_{k-1} \iff L \in \exists\Pi_{k-1} \iff L \in \Sigma_k$

Polynomial-time Hierarchy for Promise Problems

Applications

Application to Quantum computing

$H_{Promise\text{-}BQP}$ is a collapsible hierarchy for classifying quantum complexity classes, which was proposed by [2]. However, the authors of this work were unable to show that their quantum polynomial hierarchy has the property of 'collapsing' whenever two distinct levels are equal, because of difficulties arising from the use of promise problems.

Conclusion

The collapsibility of H_C and the fact that C can be arbitrarily chosen, gives us a powerful tool to generalize the hierarchy for the new collections of complexity classes that have emerged in the world of computer science in recent years.

Thanks for Your attention!

References I

Chirag Falor, Shu Ge, and Anand Natarajan.
A collapsible polynomial hierarchy for promise problems, 2023.

Sevag Gharibian, Miklos Santha, Jamie Sikora, Aarthi Sundaram, and Justin Yirka.
Quantum generalizations of the polynomial hierarchy with applications to QMA(2).
CoRR, abs/1805.11139, 2018.

Oded Goldreich.
On Promise Problems: A Survey, pages 254–290.
Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

Juraj Hromkovic.
Algorithmics for hard problems - introduction to combinatorial optimization, randomization, approximation, and heuristics.
Springer, 2001.

Jonathan Katz.
Notes on complexity theory, lecture 13, February 2012.

References II

Isaac L. Chuang Michael A. Nielsen.
Quantum Computation and Quantum Information: 10th Anniversary Edition.
Cambridge University Press, 10 anv edition, 2011.

JLuca Trevisan.
Computational complexity handout 5, 2010.