Quantum Query Complexity

IonQ Investor Pack

Optimization: Logistics

Problem

As an illustrative example among many parcel services: a UPS driver makes an average of 135 deliveries daily.[1] The number of possible routes they could take is so large, it has 227 digits. It would take a classical computer longer than the age of the universe to calculate the truly optimal route for just one driver. UPS would like to do this for more than 66,000 routes, daily.[1]

Solution

UPS estimates that their current software, which only provides approximately optimal routes, saves the company 100 million miles each year, at a cost savings of approximately $250 million per year.[2] With 1000 algorithmic qubits, a quantum computer could find truly optimal routing, saving additional millions.

Algorithmic Qubits

1000

Year Enabled

2028

IonQ Projects

Volkswagen

Successfully ran a broadly-applicable optimization problem (Binary Paint Shop) on IonQ hardware[3]

International Telecom Firm

Projects focusing on telecommunications network and logistics optimization

1 UPS To Enhance ORION With Continuous Delivery Route Optimization, UPS Pressroom (2020)

2 ORION Backgrounder, UPS Pressroom (2020)

3 Beating classical heuristics for the binary paint shop problem with the quantum approximate optimization algorithm, arXiv:2011.03403 (2020)

Note UPS is used as an illustrative example only. IonQ is not currently engaged with UPS as a customer.

Query Complexity

Last time we talked about how a generic search algorithm can find a satisfying assignment in a formula.

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$$

We do not think that quantum computers can do this efficiently.

The reasons come from a model called quantum query complexity.

Query Complexity

Many of the algorithms we have discussed have been formulated in the query model.

Instead of explicitly being given an input, we are given the ability to query values of that input with an "oracle".

We only care about the number of times the algorithm queries this oracle.

Operations in between queries can be of arbitrary complexity.

Query Complexity

Often times query algorithms end up being efficient anyway.

We saw an example of this with Shor's factoring algorithm.

We studied period finding in the query model and then plugged in an efficient circuit for the function

$$f(a) = x^a \bmod N$$

to get an efficient algorithm overall.

Advantages of Query Complexity

A motivation for query complexity is that we are unable to show classical or quantum lower bounds on circuit complexity.

We are able to show often tight lower bounds on query complexity.

- This allows rigorous separation of the classical and quantum complexity of certain problems.

- This allows us to give evidence that some problems are hard even for quantum computers.

Model Definition

Query Complexity

Setup: A function $f : \{0,1\}^n \to \{0,1\}$ is known to the algorithm.

The algorithm is given oracle access to some unknown input $x \in \{0,1\}^n$.

The algorithm can only learn about x by making queries of the form "what is x_i ?".

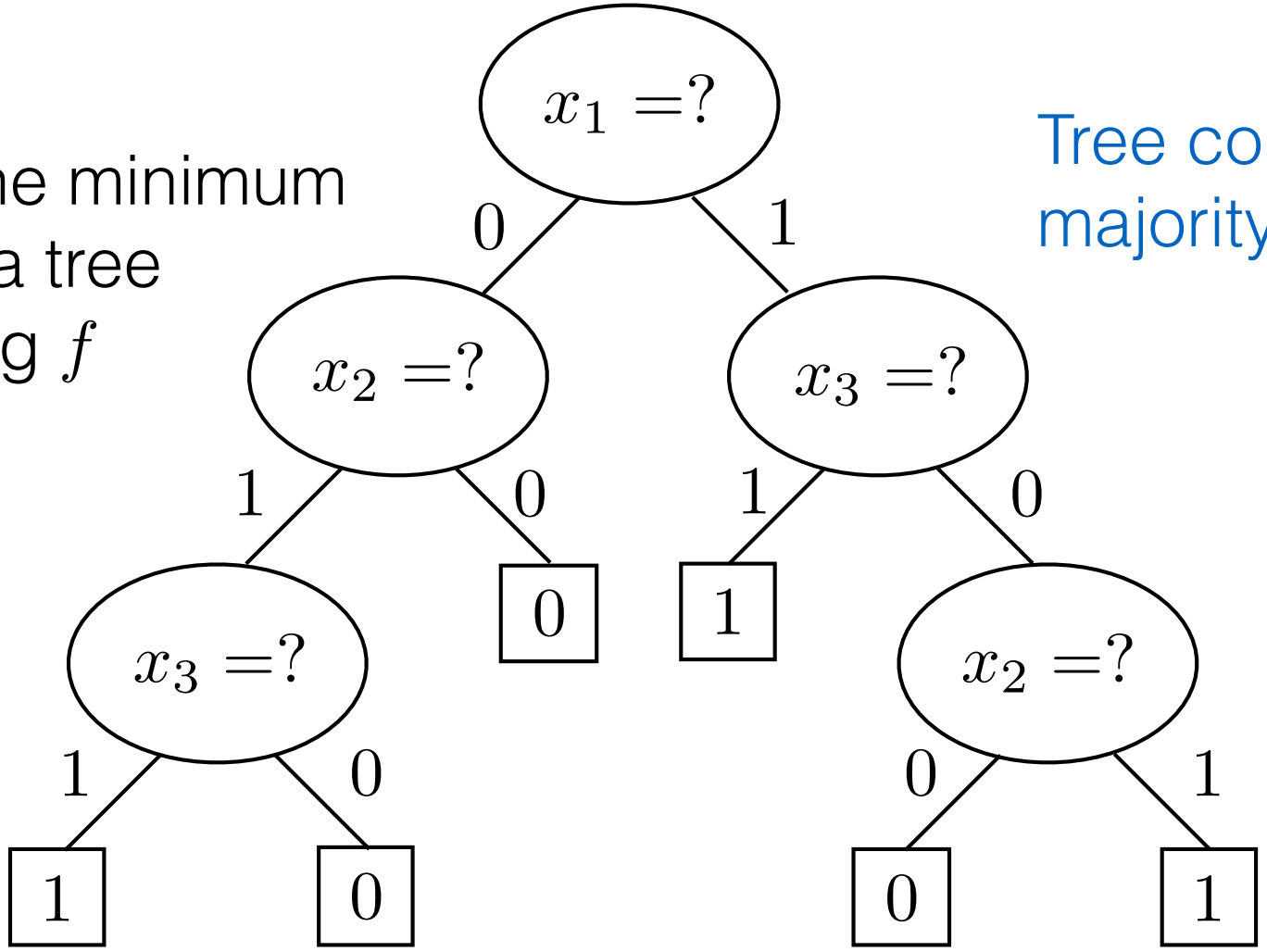
How many such queries are needed in order to determine $f(x)$?

Usually interested in the worst-case x.

Models of Query Complexity

A deterministic algorithm is described by a decision tree.

$D(f)$ is the minimum depth of a tree computing f



Tree computing majority on 3 bits

Poll

Models of Query Complexity

A randomized algorithm $\mathcal{A}$ is a probability distribution over decision trees.

$\mathcal{A}(x)$ is a random variable: pick a tree and evaluate it.

$\mathrm{cost}(\mathcal{A}, x)$ is the expected number of queries on input x

Different notions of "computing" a function.

Zero-error: every decision tree in the support computes f

Bounded-error: $\Pr[\mathcal{A}(x) = f(x)] \geq \frac{9}{10}$ for all inputs x

Example

Let's go back to the majority function on 3 bits.

Here's a zero-error randomized algorithm:

Randomly choose two positions and query them.

If they are the same, output that value.

Else, query the third bit and output it.

On input 110, with probability 1/3 we make 2 queries and with probability 2/3 we make 3 queries.

Cost is $\frac{2}{3} + 2$ (by symmetry this input is the worst case)

Models of Query Complexity

A randomized algorithm $\mathcal{A}$ is a probability distribution over decision trees.

$\mathcal{A}(x)$ random variable: pick tree and evaluate it

$\mathrm{cost}(\mathcal{A}, x)$ is the expected number of queries on input x

$$R_0(f) = \min_{\mathcal{A} \text{ zero-error}} \max_x \mathrm{cost}(\mathcal{A}, x)$$

$$R(f) = \min_{\mathcal{A} \text{ bounded-error}} \max_x \mathrm{cost}(\mathcal{A}, x)$$

Quantum Oracle

Recall the phase oracle O_x for a string $x \in \{0,1\}^n$:

$$O_x|i\rangle|b\rangle = (-1)^{x_i \cdot b}|i\rangle|b\rangle$$

$i \in \{1, \ldots, n\}$

$b \in \{0,1\}$

When $b = 0$ we call this a null query.

Usually we use a more succinct version of the phase oracle.

$$\tilde{O}_x|i\rangle = \begin{cases} |i\rangle & \text{if } i = 0 \\ (-1)^{x_i}|i\rangle & \text{otherwise} \end{cases}$$

$i \in \{0, \ldots, n\}$

This is what I'll use from now on (and drop the tilde).

Question

Why do we need to allow the null query?

Quantum Model

In addition to the query register we allow the algorithm a workspace register.

Basis states for state of the algorithm:

$$|i\rangle|t\rangle$$

query register $i \in \{0, \ldots, n\}$

workspace register $t \in \{0,1\}^m$

The algorithm begins in the state $|0\rangle|0^m\rangle$.

Quantum Model

The algorithm begins in the state $|0\rangle|0^m\rangle$.

We then interleave applying the oracle and arbitrary unitaries.

A T query algorithm is determined by unitaries $U_0, \ldots, U_T$ and a two-outcome projective measurement $\{\Pi_0, \Pi_1\}$.

On input x after t queries the state of the alg. is

$$|\psi_x^t\rangle = U_t O_x \cdots U_1 O_x U_0 |0\rangle|0^m\rangle$$

Quantum Model

On input x after t queries the state of the alg. is

$$|\psi_x^t\rangle = U_t O_x \cdots U_1 O_x U_0 |0\rangle |0^m\rangle$$

The probability the alg. outputs 1 is $\|\Pi_1 |\psi_x^T\rangle\|^2$.

The algorithm is successful for f iff

- $\|\Pi_1 |\psi_x^T\rangle\|^2 \geq 3/4$ for every $x \in f^{-1}(1)$
- $\|\Pi_1 |\psi_x^T\rangle\|^2 \leq 1/4$ for every $x \in f^{-1}(0)$

Quantum Query Complexity

The algorithm is successful for f iff

- $\|\Pi_1|\psi_x^T\rangle\|^2 \geq 3/4$ for every $x \in f^{-1}(1)$
- $\|\Pi_1|\psi_x^T\rangle\|^2 \leq 1/4$ for every $x \in f^{-1}(0)$

The quantum query complexity $Q(f)$ of f is the minimum number of queries made by a quantum algorithm successful for f .

Separations

Relationships

We have now introduced 3 different models

- Deterministic query complexity $D(f)$.
- Bounded-error randomized query complexity $R(f)$.
- Bounded-error quantum query complexity $Q(f)$.

What are the relationships between these measures?

An important distinction here is partial vs. total functions.

Partial vs. Total

A total function is one whose domain is all of $\{0,1\}^n$.

A partial function $f : S \to \{0,1\}$ for $S \subset \{0,1\}^n$ is only defined on a strict subset of $\{0,1\}^n$.

Partial function examples:

Deutsch-Josza: Only defined on $\{0^n\} \cup \{x : |x| = n/2\}$

Simon: Only defined on $x \in \{0,1\}^n$ that are the truth table of a function f satisfying Simon's condition.

Partial functions

Deutsch-Josza: $Q(f) = 1, R(f) = O(1), D(f) = \Omega(n)$

This essentially settles the $Q(f)$ vs. $D(f)$ question.

Simon's problem:

$$Q(f) = O(\log n), R(f) = \Theta(\sqrt{n}), D(f) = \Theta(\sqrt{n})$$

This is a nearly optimal separation between $Q(f)$ and $R(f)$.

k-Forrelation: $Q(f) = q, R(f) = \tilde{\Omega}(n^{1-1/(2q)})$

This is optimal!

arXiv:1411.5729

arXiv:2008.07003

Partial functions

k-Forrelation: For any constant q gives an example where

$$Q(f) = q, R(f) = \tilde{\Omega}(n^{1-1/(2q)})$$

arXiv:2008.07003

QIP 2021

This separation is optimal!

arXiv:1411.5729

Total Functions

For total functions the situation is vastly different.

For every total f

$$D(f) = O(Q(f)^4)$$

arXiv:2010.12629

QIP 2021

There exists a total f with

$$D(f) = \Omega(Q(f)^4)$$

arXiv:1506.04719

Total Functions

For total functions the situation is vastly different.

For every total f

$$R(f) = O(Q(f)^4)$$

arXiv:2010.12629

There exists a total f with

$$R(f) = \Omega(Q(f)^{3-o(1)})$$

arXiv:1511.01937

arXiv:2008.07003

Still a gap here. The conjecture is 3 is the right exponent.

Lower Bounds

OR function

The example we will focus on is the OR function.

Let $\mathrm{OR}_N(x) : \{0,1\}^N \to \{0,1\}$ be

$$\mathrm{OR}_N(x) = \begin{cases} 0 & \text{if } x = 0^N \\ 1 & \text{otherwise} \end{cases}$$

Theorem: $Q(\mathrm{OR}_N) = \Omega(\sqrt{N})$.

A quantum algorithm cannot efficiently solve SAT in a generic way.

Polynomial and Adversary

There are two main techniques for showing quantum query complexity lower bounds, the polynomial method and the adversary method.

We will see how to show a lower bound on OR_N using both of these methods.

Boolean Functions

Boolean functions are the central object in query complexity:

$$f : \{0,1\}^n \to \{0,1\}$$

Query complexity leads to considering all sorts of complexity measures of Boolean functions (see the book Analysis of Boolean Functions by Ryan O'Donnell.)

We are going to switch between several different representations of a Boolean function.

Boolean Functions

$$f : \{0,1\}^n \to \{0,1\}$$

$$f_{\pm} : \{-1,+1\}^n \to \{-1,+1\}$$

Usually we map 0 to 1 and 1 to -1.

$$f_{\pm}(1-2x) = 1-2f(x) \quad \text{for all} \quad x \in \{0,1\}^n$$

This transformation doesn't change the complexity measures we are interested in.

We will freely switch between them.

Polynomial Method

Polynomials

Now we will look at the representation of a Boolean function as a polynomial.

For this it will be convenient to use the "sign" version of the function.

$$f : \{-1, +1\}^n \to \{-1, +1\}$$

Typically we think of -1 as representing "true".

Parity functions

The sign representation makes parity functions very easy to express.

input	output
(1,1)	1
(1,-1)	-1
(-1,1)	-1
(-1, -1)	1

Output -1 iff the number of input -1 is odd

This is given by the multilinear polynomial $x_1 x_2$.

The highest power of any variable is 1.

Parity functions

In general for $S \subseteq \{1, \ldots, n\}$ we can define the function

$$\chi_S : \{-1, 1\}^n \to \{-1, 1\}$$

$$\chi_S(x) = \prod_{i \in S} x_i$$

Output -1 iff the number of input -1 in S is odd.

These are again multilinear polynomials.

Where have we seen these before?

Hadamard Matrix

$$\begin{array}{c} \\ 00 \\ 01 \\ 10 \\ 11 \end{array} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}$$

$$\begin{array}{cccc} \chi_{\emptyset} & \chi_{\{2\}} & \chi_{\{1\}} & \chi_{\{1,2\}} \end{array}$$

The set of all 2^n parity functions form an orthogonal basis for $\mathbb{R}^{2^n}$.

Every Boolean function can be uniquely expressed as a multilinear polynomial.

AND function

Let's look at the function $\mathrm{AND}_2 : \{-1, +1\}^2 \to \{-1, +1\}$.

input	output
(1,1)	1
(1,-1)	1
(-1,1)	1
(-1, -1)	-1

How can you express this function as a polynomial?

input	output
(1,1)	1
(1,-1)	1
(-1,1)	1
(-1, -1)	-1

It is equivalent to expressing the truth table as a linear combination of the parity functions.

$$\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} - \frac{1}{2}\begin{bmatrix} 1 \\ -1 \\ -1 \\ 1 \end{bmatrix}$$

$$\mathrm{AND}_2(x_1, x_2) = \frac{1}{2}\left(1 + x_1 + x_2 - x_1 x_2\right)$$

AND function

Let's look at the AND function on 2 bits.

$$\mathrm{AND}_2(x_1, x_2) = \frac{1}{2}\left(1 + x_1 + x_2 - x_1 x_2\right)$$

We can obtain this expression by applying the Hadamard matrix with slightly different normalization

$$\frac{1}{4}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix} = \frac{1}{2}\begin{bmatrix} 1 \\ 1 \\ 1 \\ -1 \end{bmatrix}$$

Fourier Coefficients

In general, for $f : \{-1,1\}^n \to \{-1,1\}$

$$f(x) = \sum_{S \subseteq [n]} \hat{f}_S \chi_S(x)$$

where $\hat{f}_S = \frac{1}{2^n} \langle \chi_S, f \rangle$.

In other words, $H^{\otimes n} f / \sqrt{2^n}$ gives the vector of all Fourier coefficients.

Polynomial Degree

$$f(x) = \sum_{S \subseteq [n]} \hat{f}_S \chi_S(x)$$

The polynomial degree of f is $\max\{|S| : \hat{f}_S \neq 0\}$.

Examples:

$$\mathrm{AND}_2(x_1, x_2) = \frac{1}{2}(1 + x_1 + x_2 - x_1 x_2)$$

degree 2

$$\mathrm{MAJ}_3(x_1, x_2, x_3) = \frac{1}{2}(x_1 + x_2 + x_3 - x_1 x_2 x_3)$$

degree 3

Approximating Polynomial

We are now going to talk about polynomials that approximate functions.

This is what is needed for lower bounds on bounded-error quantum query complexity.

For this I am going to switch to using $\{0,1\}$ as the range.

$$f = (1 - f_{\pm})/2$$

This change does not affect the degree.

Approximating Polynomial

A polynomial $p(x_1, \ldots, x_n)$ is an ε-error approximating polynomial iff $|f(x) - p(x)| \leq \varepsilon$ for all $x \in \{-1, +1\}^n$.

We let $\deg_\varepsilon(f)$ be the minimum degree of an ε-error approximating polynomial for f.

This can be smaller than $\deg(f)$.

AND Example

$$p(x_1, x_2) = \frac{1}{3}(1 - x_1 - x_2)$$

is a $\frac{1}{3}$ -error approximating polynomial for

$$\mathrm{AND}_2 : \{-1, 1\}^2 \to \{0, 1\}$$

input	output	p
(1,1)	0	-1/3
(1,-1)	0	1/3
(-1,1)	0	1/3
(-1, -1)	1	1

Polynomial Method

Theorem: If there is a T query quantum algorithm to compute $f : \{-1, 1\} \to \{0, 1\}$ with error at most $1/3$ then $\deg_{1/3}(f) \leq 2T$.

quant-ph/9802049

Idea: Look at the function $p(x) = \Pr[\mathcal{A}(x) = 1]$ where $\mathcal{A}$ is a $1/3$-error quantum algorithm for f.

This function is a $1/3$-error approximating polynomial for f.

If $\mathcal{A}$ makes T queries we will show the degree of p is at most $2T$.

The key claim to make this work is the following.

Suppose that after t queries on input $x \in \{-1,1\}^n$ the state of the algorithm is

$$|\psi_x^t\rangle = \sum_{i,w} \alpha_{i,w}(x)|i\rangle|w\rangle$$

where each $\alpha_{i,w}(x)$ is a polynomial of degree $\leq t$.

If we now make a query the state becomes

$$O_x|\psi_x^t\rangle = \sum_{i,w}(x_i \cdot \alpha_{i,w}(x))|i\rangle|w\rangle$$

Each amplitude is now a degree $\leq t+1$ polynomial.

Suppose that after t queries on input $x \in \{-1,1\}^n$ the state of the algorithm is

$$|\psi_x^t\rangle = \sum_{i,w} \alpha_{i,w}(x)|i\rangle|w\rangle$$

where each $\alpha_{i,w}(x)$ is a polynomial of degree $\leq t$.

If we perform a unitary U then

$$U|\psi_x^t\rangle = \sum_{i,w} \beta_{i,w}(x)|i\rangle|w\rangle$$

Each $\beta_{i,w}$ is a linear combination of the $\alpha_{i,w}$. The degree does not increase.

Recall the choice of unitary does not depend on x.

Start of Algorithm

The algorithm begins in the state $|0\rangle|0^m\rangle$ for every x.

Initially, each $\alpha_{i,w}(x)$ has degree 0.

This shows by induction that after T queries the state of the alg. can be written as

$$|\psi_x^T\rangle = \sum_{i,w} \alpha_{i,w}(x)|i\rangle|w\rangle$$

where each $\alpha_{i,w}(x)$ is a degree $\leq T$ polynomial.

Measurement

$$|\psi_x^T\rangle = \sum_{i,w} \alpha_{i,w}(x)|i\rangle|w\rangle$$

where each $\alpha_{i,w}(x)$ is a degree $\leq T$ polynomial.

At the end of the algorithm we make a 2-outcome projective measurement $\{\Pi_0, \Pi_1\}$.

The probability the algorithm outputs 1 on input x is

$$p(x) = \|\Pi_1|\psi_x^T\rangle\|^2$$

which is a degree $2T$ polynomial.

Summary

We have a method to show lower bounds on quantum query complexity

$$Q_{1/3}(f) \geq \frac{\deg_{1/3}(f)}{2}$$

Now we will use this method to show that Grover's algorithm is tight

$$Q_{1/3}(\mathrm{OR}_n) = \Omega(\sqrt{n})$$

Application to OR

OR Application

It is not very easy to lower bound approximate polynomial degree in general.

But OR_n is a very nice function because it is symmetric.

The output only depends on the number of ones in the input, not on where those ones are located.

We completely understand the approximate polynomial degree of such functions.

Symmetrization

Let $f : \{0,1\}^n \to \{0,1\}$ be a symmetric function.

Let $p(x_1, \ldots, x_n)$ be an ε-error approximating polynomial for f of degree d.

If we symmetrize p it will still be an ε-error approximating polynomial for f of degree d.

$$p_{\mathrm{sym}}(x_1, \ldots, x_n) = \frac{1}{n!} \sum_{\pi \in S_n} p(\pi(x_1, \ldots, x_n))$$

Note that f is unchanged under symmetrization.

Symmetrization: Example

Let's think about $\mathrm{AND}_2 : \{0,1\}^2 \to \{0,1\}$ again.

$$p(x_1, x_2) = \frac{6}{20}x_1 + \frac{7}{20}x_2$$

provides a $7/20$ approximation.

input	output	p
(0,0)	0	0
(0,1)	0	6/20
(1,0)	0	7/20
(1, 1)	1	13/20

$$p_{\mathrm{sym}}(x_1, x_2) = \frac{1}{2}\left(p(x_1, x_2) + p(x_2, x_1)\right)$$

$$= \frac{13}{40}\left(x_1 + x_2\right)$$

The error on each Hamming weight is the average of the previous errors.

Symmetrization

$$p_{\mathrm{sym}}(x_1, \ldots, x_n) = \frac{1}{n!} \sum_{\pi \in S_n} p(\pi(x_1, \ldots, x_n))$$

In p_{sym} the coefficient of a monomial depends only on the degree of that monomial.

If p has degree d there are numbers $c_0, \ldots, c_d$ such that

$$p_{\mathrm{sym}}(x_1, \ldots, x_n) = \sum_{k=0}^{d} c_k \sum_{\substack{S \\ |S|=k}} \prod_{i \in S} x_i$$

Symmetrization

$$p_{\mathrm{sym}}(x_1, \ldots, x_n) = \sum_{k=0}^{d} c_k \underbrace{\sum_{\substack{S \\ |S|=k}} \prod_{i \in S} x_i}_{\text{focus here}}$$

Example: For $x = 1111000$ and $k = 2$ the sets S for which $\prod_{i \in S} x_i = 1$ are

$$S = \{1,2\}, \{1,3\}, \{1,4\}, \{2,3\}, \{2,4\}, \{3,4\}$$

In general:

$$\sum_{\substack{S \\ |S|=k}} \prod_{i \in S} x_i = \binom{|x|}{k}$$

Symmetrization

$$p_{\mathrm{sym}}(x_1, \ldots, x_n) = \sum_{k=0}^{d} c_k \binom{|x|}{k}$$

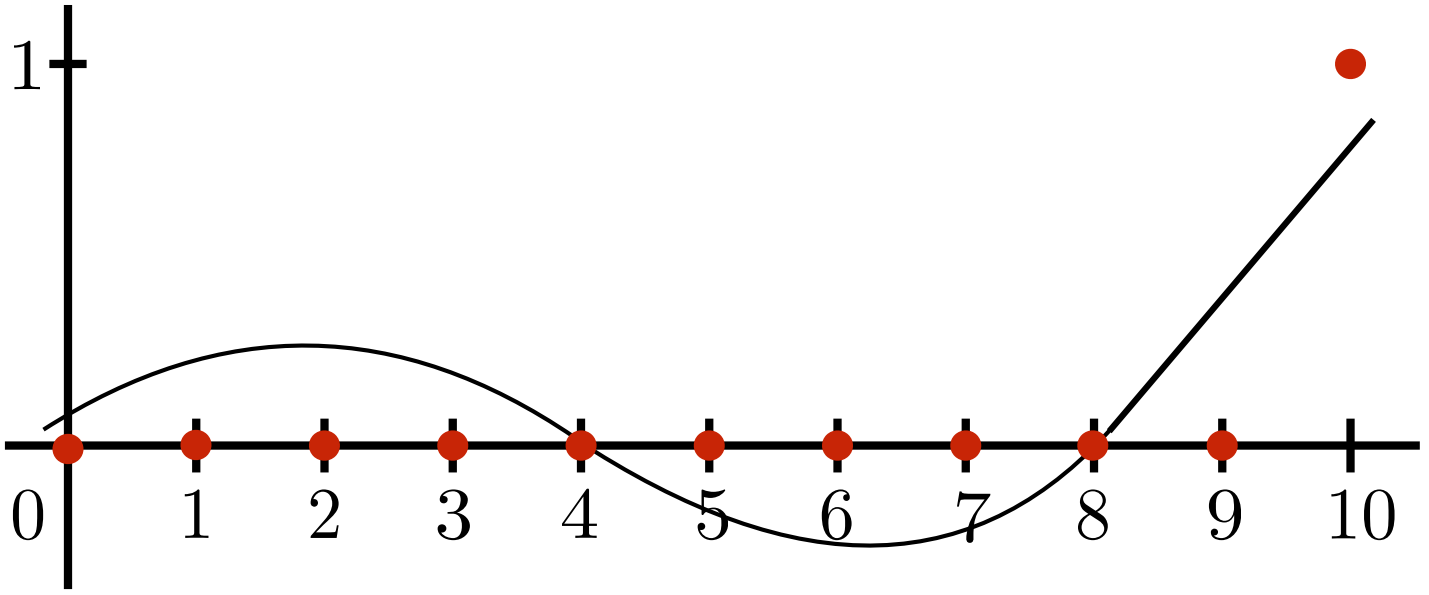
Now we define a univariate polynomial $P(z)$

$$P(z) = \sum_{k=0}^{d} \frac{c_k}{k!} z(z-1)\cdots(z-k+1)$$

This polynomial has degree d and $P(|x|) = p_{\mathrm{sym}}(x)$ for all $x \in \{0,1\}^n$.

Now we can show lower bounds on the degree of a univariate polynomial.

AND

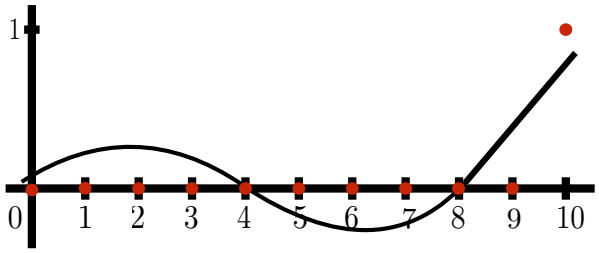


For $n = 10$ our polynomial needs to be in $[-1/3, 1/3]$ for $z = 0, \ldots, 9$ and in $[2/3, 4/3]$ for $z = 10$.

The polynomial has to "jump" from near zero at $n - 1$ to near one at n.

Markov Inequality

In the 1890s Andrey Markov proved



$$\max_{x\in[0,n]} |P'(x)| \leq \frac{\deg(P)^2}{n} \max_{x\in[0,n]} |P(x)|$$

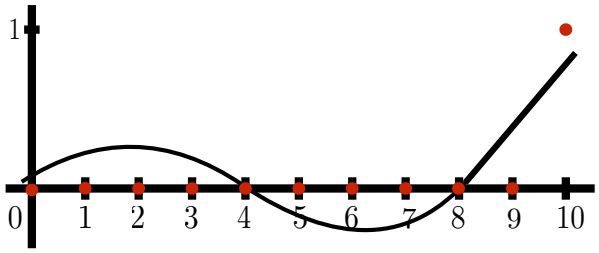
If P comes from a $1/3$-error approximating polynomial for AND_n then $P(n-1) \leq 1/3$ and $P(n) \geq 2/3$.

This means $\max_{x\in[0,n]} |P'(x)| \geq 1/3$.

If $\max_{x\in[0,n]} |P(x)| \leq 2$ then $\deg(P) \geq \sqrt{\frac{n}{6}}$.

Lower bound for AND

In the 1890s Andrey Markov proved



$$\max_{x\in[0,n]} |P'(x)| \leq \frac{\deg(P)^2}{n} \max_{x\in[0,n]} |P(x)|$$

If $\max_{x\in[0,n]} |P(x)| = h > 2$ then $\max_{x\in[0,n]} |P'(x)| \geq h - 1/3$.

In this case $\deg(P) \geq \sqrt{\frac{5n}{6}}$.

Either way, $\deg(P) = \Omega(\sqrt{n})$.

Summary

We have shown that it takes $\Omega(\sqrt{n})$ quantum queries in order to compute AND_n with success prob. $2/3$.

The same lower bound applies to OR_n because

$$\mathrm{OR}_n(x_1, \ldots, x_n) = \neg \mathrm{AND}_n(\neg x_1, \ldots, \neg x_n)$$

This shows Grover's algorithm is optimal!

Weaknesses

The polynomial method does not show a tight quantum query lower bound for every function.

We did not use unitarity of operations in the proof!

The polynomial method lower bounds a stronger model where arbitrary linear transformations are allowed in between queries.

There is a strengthening of the polynomial method that takes this into account to characterize quantum query complexity [ABP19].