Simon's Problem

PHYS/CSCI 3090

Prof. Alexandra Kolla

Alexandra.Kolla@Colorado.edu
ECES 122

Prof. Graeme Smith

Graeme.Smith@Colorado.edu
JILA S326

https://home.cs.colorado.edu/~alko5368/indexCSCI3090.html

Come see us!

- Alexandra Kolla/ Graeme Smith: Friday 3:00-4:00 pm, JILA X317.
- Ariel Shlosberg: Tu/Th 2:00-4:00pm, DUANG2B90 (physics help room)
- Steven Kordonowy: Th 11am-12pm, ECAE 124.
- Matteo Wilczak: Wednesday, 1-2pm, DUANG2B90 (physics help room)

Midterm 1

- Midterm 1 in class in 2 days (Feb 12)
- 2pm in JILA X325, which is right next to X317

Last Class

- Bernstein-Vazirani
- Start of Simon’s

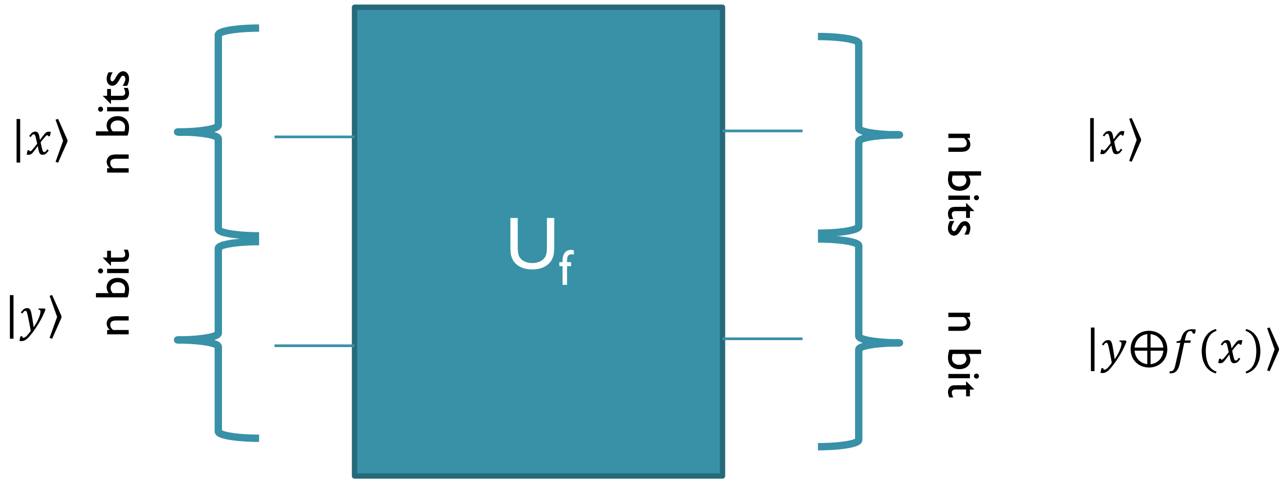
Today

- Simon’s problem
- While Bernstein Vazirani gets linear speedup on quantum computer, we can achieve exponential speedup for Simon’s problem

Simon's problem

- One is told that f is periodic under bitwise modulo-2 addition, $\mathrm{f}(\mathrm{x} \oplus a) = f(x), for\ all\ x$
- The problem is to find the period a.
- Precursor to Shor's factoring, where we are interested in functions that are periodic under ordinary addition (decimal).

The setup



Steps:

- Prepare the input register in uniform superposition
- Apply U_f
- Measure output register

The Algortihm

- 1) Prepare: $(H^{\otimes n} \otimes \mathrm{I})\ |0\rangle_n \quad |0\rangle_n = \frac{1}{2^{n/2}} \sum_{0<x\leq 2^n} |x\rangle_n\ |0\rangle_n$

- 2) Orace: $\mathrm{U_f}(\frac{1}{2^{n/2}} \sum_{0<x\leq 2^n} |x\rangle_n\ |0\rangle_n) = \frac{1}{2^{n/2}} \sum_{0<x\leq 2^n} |x\rangle_n\ |f(x)\rangle_n$

- 3)Measure output register: If I get some value of f, say f(x_0), then input is $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle)$

The Algortihm

3)Measure output register: If I get some value of f, say f(x_0), then input is $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle)$

The Algortihm, cont

- 3) Measure output register: If I get some value of f, say f(x$_o$), then input is $\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle)$
- 4)Apply $H^{\otimes n}$ to input register

$$Recall\text{: } H^{\otimes n}\,|x\rangle_n \quad = \frac{1}{2^{n/2}}\sum_{y=0}^{2^n-1}(-1)^{y\cdot x}\,|y\rangle_n$$

$$H^{\otimes n}\,\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle) =$$

$$\frac{1}{2^{(n+1)/2}}\sum_{y=0}^{2^n-1}((-1)^{y\cdot x_0} + (-1)^{y\cdot(x_0 \oplus a)})|y\rangle_n$$

The Algortihm, cont

$$H^{\otimes n}\ \frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle)=$$
$$\frac{1}{2^{(n+1)/2}}\sum_{y=0}^{2^n-1}((-1)^{y\cdot x_0} + (-1)^{y\cdot(x_0\oplus a)})|y\rangle_n$$

- Since $(-1)^{y\cdot(x_0\oplus a)} = (-1)^{y\cdot x_0}(-1)^{y\cdot a}$, the coefficient of $|y\rangle$ is zero if $y \cdot a = 1$ and $2(-1)^{y\cdot x}$ if $y \cdot a = 0$
- State is: $\frac{1}{2^{(n-1)/2}}\sum_{y\cdot a=0}(-1)^{y\cdot x_0}\,|y\rangle_n$,
- Only the y's such that $a \cdot y = 0$ survive!
- If we measure the input register, we learn with equal probability any of the values of y such that $a \cdot y = 0$.

Analysis of the Algorithm

- With each invocation of U_f, we learn a random y satisfying
$$a \cdot y = \sum_{i=0}^{n-1} y_i a_i = 0 \; mod \; 2 \; .$$
- If we call U_f m times, we learn m independently selected random numbers y with this property.
- Need to do some math to see how this helps.

Analysis of the Algorithm

- Definition: a set of vectors $y^{(1)}, \ldots, y^{(m)}$ is linearly independent, if there is no subset of those vectors such that $y^{(i_1)} \oplus \cdots \oplus y^{(i_j)} = 0 \; mod \; 2$

Linear independence

Assume I have m linear equations (mod 2) of the form $\sum_{i=0}^{n-1} y_i^{(k)} a_i = 0 \, mod \, 2$. For m different vectors $y^{(1)}, \dots, y^{(m)}$. Assume, moreover, that the $y^{(k)}$ are all linearly independent. What does m need to be in order to completely determine a?

A) n

B) 1

C) $n - 1$

D) n^2

Analysis of the Algorithm

- With each invocation of U_f, we learn a random y satisfying $a \cdot y = \sum_{i=0}^{n-1} y_i a_i = 0 \; mod \; 2$.
- If we call U_f m times, we learn m independently selected random numbers y with this property.
- We have to invoke the subroutine enough times to give us high probability of coming up with n-1 linearly independent y.

Analysis of the Algorithm

- Let $S_i = Span\{y^{(1)}, y^{(2)}, \dots, y^{(i)}\}$ and D_i the dimension of S_i.

How many elements?

Let $S_i = Span\{y^{(1)}, y^{(2)}, \ldots, y^{(i)}\}$ and D_i the dimension of S_i after the i-th iteration. Assume D_i=k. How many elements does S_i have? In other words, what is $|S_i|$?

A) 2^n

B) 2^k

C) k

D) n

Conditional Probability

Let $S_i = Span\{y^{(1)}, y^{(2)}, \dots, y^{(i)}\}$ and D_i the dimension of S_i after the i-th iteration.
What is $P(D_{i+1} = k + 1 | D_i = k)$?

A) $\frac{2^n - |S_i|}{2^n}$

B) 1

C) $\frac{n-k}{2^n}$

D) $\frac{n-k}{n}$

Conditional Probability II

Let $S_i = Span\{y^{(1)}, y^{(2)}, \ldots, y^{(i)}\}$ and D_i the dimension of S_i after the i-th iteration. What is $P(D_{i+1} = k | D_i = k)$?

A) $\frac{|S_i|}{2^n}$

B) 0

C) $\frac{k}{2^n}$

D) $\frac{k}{n}$

Analysis of the Algorithm

- Let $S_i = Span\{y^{(1)}, y^{(2)}, \ldots, y^{(i)}\}$ and D_i the dimension of S_i.
- Note that $P(D_{i+1} = k + 1 | D_i = k) = \frac{2^n - |S_i|}{2^n}$

Since each vector has probability $\frac{1}{2^n}$ of being picked.

- Also, $P(D_{i+1} = k | D_i = k) = \frac{|S_i|}{2^n}$
- There is no other value D_{i+1} can take.

Analysis of the Algorithm with coin flipping

- Let $S_i = Span\{y^{(1)}, y^{(2)}, \dots, y^{(i)}\}$ and D_i the dimension of S_i.
- $P(D_{i+1} = k | D_i = k) = \frac{|S_i|}{2^n}$
- $|S_i| = 2^k, if\ D_i = k$.
- Assume we are at iteration i, with $D_i = k$.
- Toss a coin with probability of failure $\frac{2^k}{2^n}$
- On failure, D_{i+1} remains k, on success it gets updates to k+1.

How many times to flip a coin?

Assume I have a biased coin, with probability of landing tails (failure) p, and probability of landing heads (success), 1-p.
How many times do I need to flip the coin in expectation to land heads?

A) $1 - p$

B) p

C) $\frac{1}{1-p}$

D) $\frac{1}{p}$

Analysis of the Algorithm with coin flipping

- Toss a coin with probability of failure p=$\frac{2^k}{2^n}$. Thus 1-p = $\frac{2^n-2^k}{2^n}$
- On failure, D_{i+1} remains k, on success it gets updates to k+1.
- The expected waiting time at state k (how many times do I need to flip the coin to get heads?) is $\frac{2^n}{2^n-2^k}$.
- Hence total expected time to hit n-1 is

Total expected time

The expected waiting time at state k (how many tries until we reach state k-1) is $\frac{2^n}{2^n-2^k}$. What is the total expected time of the algorithm until we reach state n-1?

A) $\frac{2^n}{2^n-2^k}$

B)$\sum_{k=0}^{n-1} \frac{2^n}{2^n-2^k}$

C)$n-1$

D) idk

Analysis of the Algorithm with coin flipping

- Toss a coin with probability of failure p=$\frac{2^k}{2^n}$.

  Thus 1-p = $\frac{2^n-2^k}{2^n}$
- On failure, D_{i+1} remains k, on success it gets updates to k+1.
- The expected waiting time at state k (how many times do I need to flip the coin to get heads?) is $\frac{2^n}{2^n-2^k}$.
- Hence total expected time to hit n-1 is

$$\sum_{k=0}^{n-1} \frac{2^n}{2^n-2^k} < \sum_{k=0}^{n-1} 2 < 2n$$

Talk on quantum supremacy later today