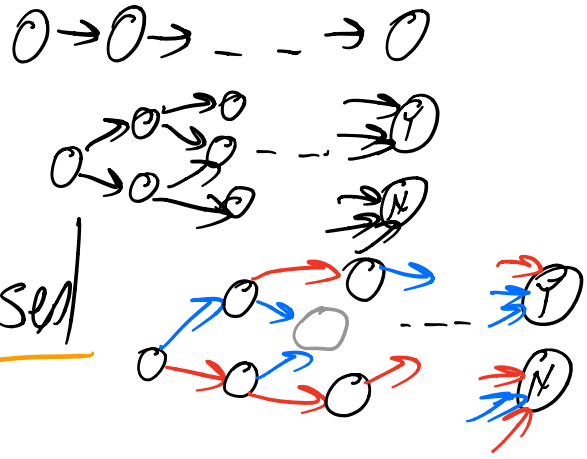
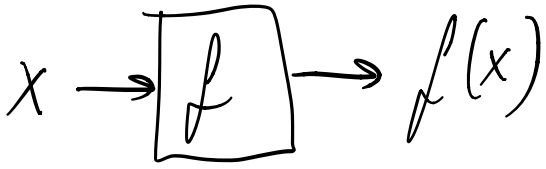
Lecture 2 - Solving hard problems with interference 1

Recap from last time

- Algorithms → Deterministic
  → Probabilistic
  → Interference-based



- Efficient algorithms - runtime is some polynomial in input size
- Oracle / black box / query model of computation



Determine if f has some property with a small number of queries (ideally a polynomial number of queries)

– The Deutsch-Jozsa problem

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

is it constant or balanced?

Deterministic: $2^{n-1}+1$ queries

Probabilistic: 3 queries

Interference: 1 query.

Interference computation = regular computation + $f: D \to \{-1,+1\}$

interf1 (f, D, N)

$\to S = \frac{1}{N} \left| \sum_{x \in D} f(x) \right|$; is $S \geq 2/3$ or $S \leq 1/3$

This lecture

– Bernstein-Vazirani problem
– Decision vs search problems
– Generalizing interf1

Some notation for binary strings

$\{0,1\}^n$ - set of binary strings of length n

$0^n = \underbrace{000\ldots0}_{\text{length } n}$ $\quad$ $1^n = \underbrace{111\ldots1}_{\text{length } n}$

For a string $x \in \{0,1\}^n$, x^i denotes the i'th bit of x; i.e. $x = x^1 x^2 \ldots x^n$, $x_i \in \{0,1\}$

For $x, y \in \{0,1\}^n$ $x \oplus y$ denotes the bitwise xor (exclusive or) operation.

$$x \oplus y = (x^1 \oplus y^1)(x^2 \oplus y^2)\ldots(x^n \oplus y^n)$$

E.g. $x = 010$

$y = 110$

$x \oplus y = 100$

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

For $x, y \in \{0,1\}^n$ $x \cdot y$ denotes the dot product

between x and y mod 2

$$x \cdot y = x^1 \cdot y^1 \oplus x^2 \cdot y^2 \oplus \cdots \oplus x^n \cdot y^n$$

a	b	$a \cdot b$
0	0	0
0	1	0
1	0	0
1	1	1

E.g. $x = 010$, $y = 110$ $\quad x \cdot y = 0 \oplus 1 \oplus 0 = 1$

The Hamming weight of a string x, denoted $|x|$ is the number of 1's in x.

E.g. $x = 110$, $|x| = 2$

$x = 111$, $|x| = 3$

$x = 000$, $|x| = 0$

The Hamming distance between string $x, y \in \{0,1\}^n$ denoted $d(x,y)$ is the smallest number of bit flips required to map x to y (or vice versa).

Equivalently $d(x,y) = |x \oplus y|$

Bernstein-Vazirani (BV) problem

We are given oracle access to $f:\{0,1\}^n \to \{0,1\}$

$f(x) = s \cdot x$, for some $s \in \{0,1\}^n$

Task: find s

E.g. $n=3$, $s=010$

$f(000)=0$ $\quad$ $f(100)=0$

$f(001)=0$ $\quad$ $f(101)=0$

$f(010)=1$ $\quad$ $f(110)=1$

$f(011)=1$ $\quad$ $f(111)=1$

Note that unlike Deutsch-Jozsa, this is a search problem rather than a decision problem. The output is an n-bit string instead of a yes/no answer.

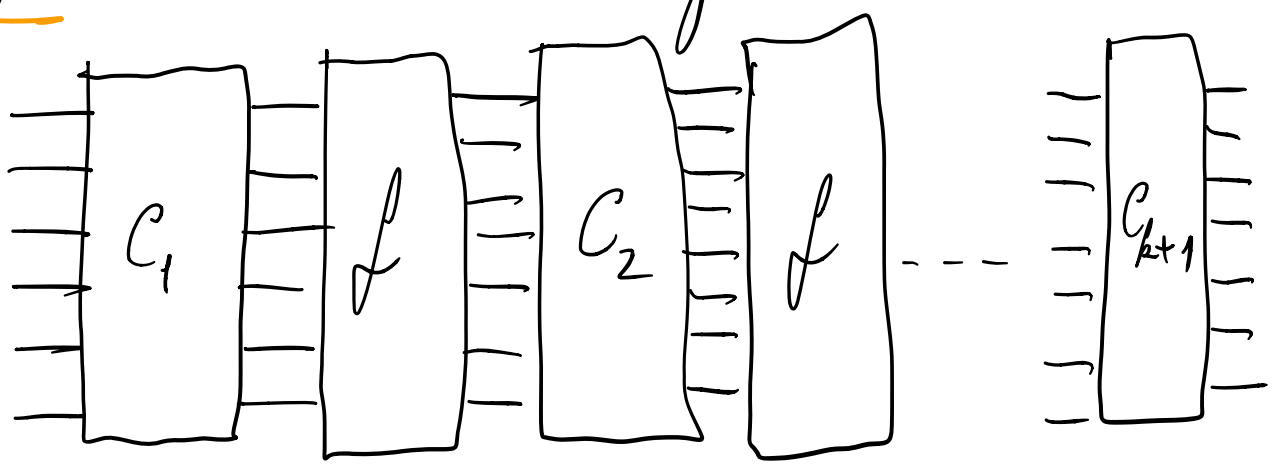
We'll have more to say about this later.

How many queries do we need to solve BV?

Deterministic case

We saw that the general form of a deterministic algorithm is something like this

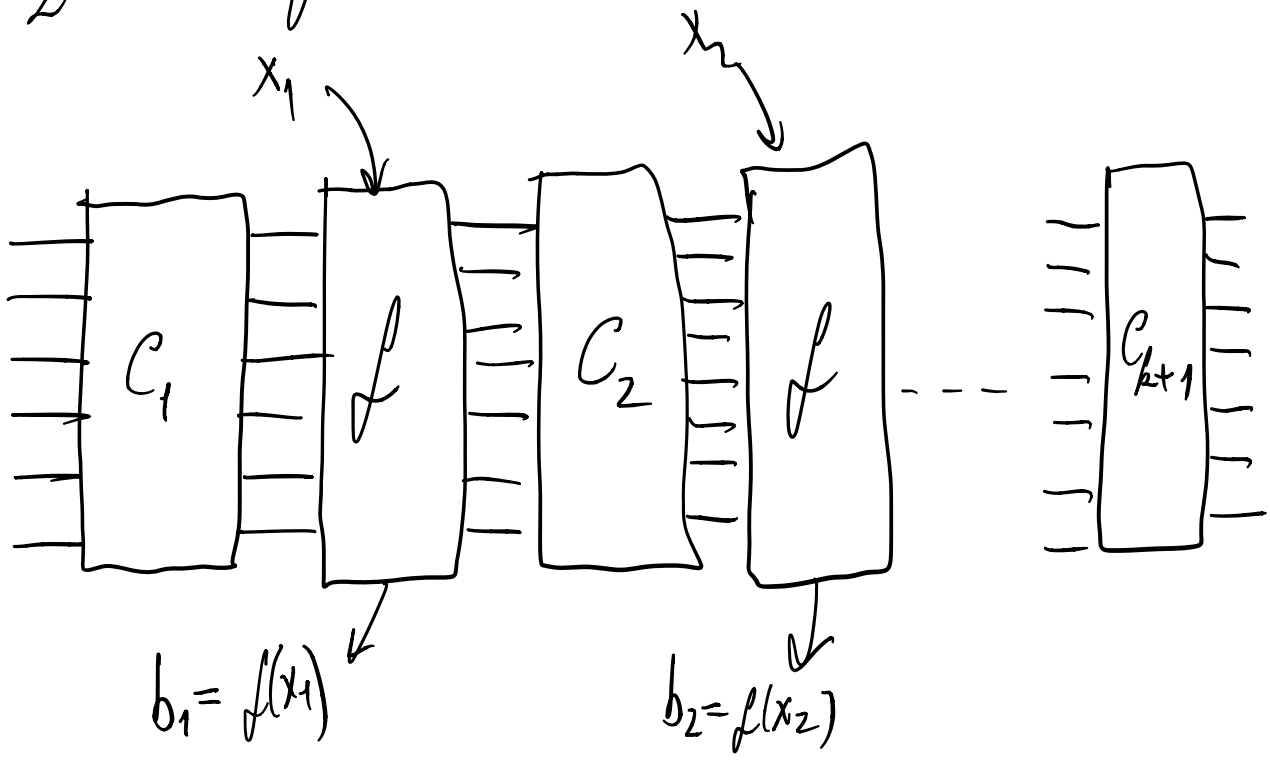
We'll now adopt a circuit view of the computation. Suppose we have an algorithm that makes k queries to f. What does such an algorithm look like?



# of calls to f is k.

Algorithm consists of oracle queries to f interspersed with non-query operations, denoted C_i, $1 \le i \le k+1$

Suppose we have such an algorithm for BV that makes k queries to f and $k < n$.



Note that the output of this algorithm is determined by the values $b_1, b_2 \ldots b_k$ and whatever other additional bits the algorithm uses (that are independent of the oracle calls)

So we have $b_1 = x_1 \cdot s$, $b_2 = x_2 \cdot s$ --- $b_k = x_k \cdot s$

for some $x_1, \ldots x_k \in \{0,1\}^n$

(note that the subscript is indicating an n-bit string whereas superscript indicates a particular bit of an n-bit string)

If the algorithm is correctly solving BV then it must produce s as its output.

But $|s| = n$ and so there are 2^n possible s values. On the other hand, we have $k < n$ b_i's. Since $k < n$, we can find an $s' \neq s$ producing the same b_i values as s. The deterministic algorithm must behave the same in both cases $\Rightarrow$ contradiction with the fact that it always outputs s correctly.

Any deterministic algorithm requires at least n queries to solve BV.

In fact we can solve it with exactly n queries.

Query the function on

$$f(00\ldots01) = 0\ldots01 \cdot s = s^1$$
$$f(00\ldots10) = s^2$$
$$\vdots$$
$$f(10\ldots0) = s^n$$

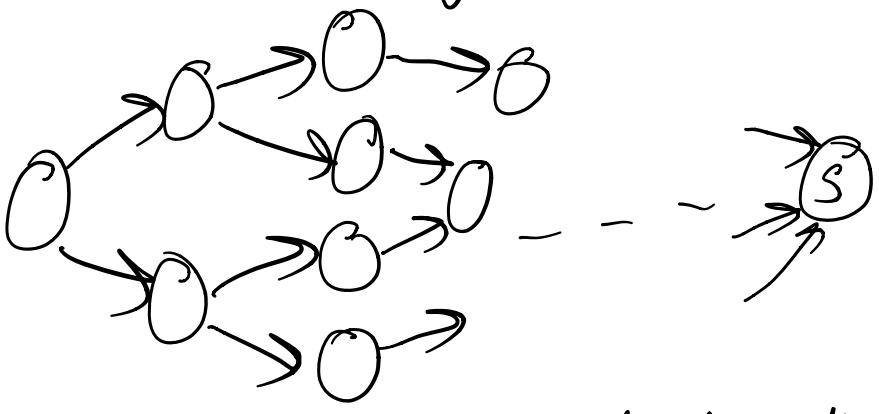
With these n queries we recover all bits in s.

Probabilistic case

The situation is similar to the deterministic case. Our algorithm now uses randomness and has to produce s with probability $\geq \frac{1}{2} + o(1)$.

If we again assume that the algorithm makes $k < n$ queries it must be that for most choices of the randomness

the algorithm decides correctly.



$\forall s \in \{0,1\}^n$, $\frac{1}{2} + o(1)$ of the paths lead to the correct s $\Rightarrow$ $\exists$ a path P that produces the correct answer on $> \frac{1}{2}$ of all inputs s

$\Rightarrow$ P finds correct s for $> 2^{n-1}$ s values

But P is a deterministic algorithm making k queries so it can output at most 2^k different s values.

Since $k < n$ this provides the desired contradiction.

Alternative proof using Shannon's source coding theorem (can't compress n bits into k bits, even probabilistically)

Interference case

Since BV is a search problem, it's not clear that using interf 1 is going to help.

We're going to instead define a more general interference function.

Thinking back at the dice example, what we want is to sample from a prob distribution resulting from interference.

The most natural way to achieve this is as follows

Given $fun: D \to \{-1, +1\}$, $Dom \subseteq D$, we're going to sample y from D according to the distribution

$$Pr(y) \sim P_y = \left| \sum_{x \in Dom} fun(x) \cdot T(x, y) \right|$$

for some $T : D \times D \to [-1, 1]$

(normally we can take $T : D \times D \to \{-1, +1\}$, but since we're removing the normalization constant $\mathcal{Z}$, we'll instead allow T to output real numbers)

Of course, we must have

$$\sum_{y \in D} \Pr(y) = 1$$

so $$\Pr(y) = \frac{P_y}{\sum_{y \in D} P_y}$$

This leads to the definition of interp2

interp2 (f, T, Dom)

Let $P_y = \left| \sum_{x \in \text{Dom}} f_{\text{un}}(x) \cdot T(x, y) \right|$

Sample y from $\Pr(y) = \frac{P_y}{\sum_y P_y}$

return y

We'll give more justification for interf2 later.
Let's solve BV first.

As you'd expect the first step is to take

$$f_{un}(x) = (-1)^{f(x)}$$

$$Dom = \{0,1\}^n$$

What should $T(x,y)$ be?

Ideally we want to choose $T(x,y)$ so that interf2 outputs s.

So we want

$$Pr(s) = 1 \sim \left| \sum_{x \in \{0,1\}^n} (-1)^{s \cdot x} \cdot T(x,s) \right|$$

Take $T(x,s) = (-1)^{s \cdot x}$

$\Rightarrow P_s = 2^n$

What about P_y for $y \neq s$

$$P_y = \left| \sum_{x \in \{0,1\}^n} (-1)^{x \cdot s} \cdot (-1)^{x \cdot y} \right|$$

$$= \left| \sum_{x \in \{0,1\}^n} (-1)^{x \cdot (s \oplus y)} \right|$$

if $y \neq s \Rightarrow |s \oplus y| > 0 \Rightarrow P_y = 0$!

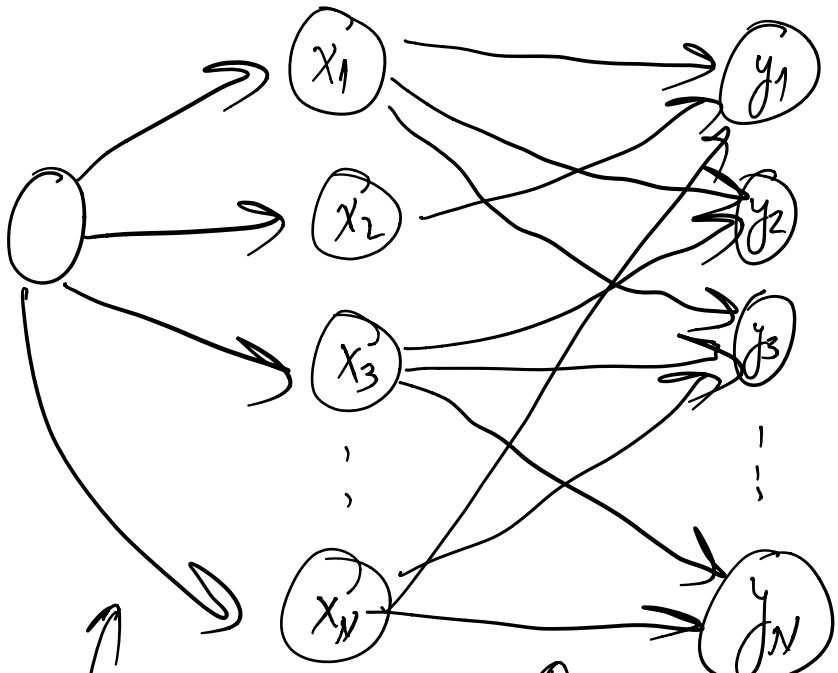
$\Rightarrow$ We get perfect constructive interference on s and perfect destructive interference on all $y \neq s$

Our interference algorithm is just

```
fun(x) = (-1)^f(x)
Dom = {0,1}^n
T(x,y) = (-1)^(x·y)
return interf2(fun, T, Dom)
```

We have just 1 query vs n queries in the prob and det. cases.

So what is $T(x, y)$?



$fun(x) = (-1)^{f(x)}$

$T(x, y)$

Note that BV is a search problem. Can we make it a decision problem? Yes!

Suppose that instead of having to return s you have to return some $g(s)$, $g: \{0,1\}^n \to \{0,1\}$

g is a hard-core predicate for s (or for f)

Computing $g(s)$ by querying f should be as hard as computing s from queries to f.

E.g.

$$g(s) = \begin{cases} 0, & |s| \bmod 3 = 0 \\ 1, & \text{otw} \end{cases}$$

The decision version allows us to make a recursive version of BV. The 2-level recursive version is

$$f(x, y) = s_x \cdot y$$

s_x chosen such that $g(s_x) = x \cdot s$, for some s. Find s! (or $g(s)$)

Classically need n^2 queries.

This can be generalised to k level recursion and you will require n^k queries to solve classically.

What about in the interference model?

Solve the homework to find out :)