A sublinear-time algorithm is an algorithm that produces an answer before looking at its whole input. One such algorithm is polling: To find out who will win an election it is usually not necessary to ask every single voter. A representative sample typically tells who the potential winner will be.

Polling, like many other sublinear-time algorithms, is a randomized procedure. Randomness makes a big difference in the power of such algorithms. Quantum computation is a more general type of computation that yields further improvements in certain cases. Although scalable quantum computers have not been built up to date (see this article for recent progress) their is a well-defined and widely accepted model that describes their potential operation.

1 Oracles, queries, and promise problems

Owing to their sequential access, Turing Machines are inadequate for describing sublinear-time algorithms. Similarly circuits of sublinear size are too restricted in power. To model such algorithms it is necessary to provide random access to the input, namely the ability to query the input at a particular location chosen by the algorithm, possibly dependent on the answers to previous queries.

An *oracle circuit* $C^?$ with oracle type $\{0,1\}^n \to \{0,1\}^m$ is a (bounded fan-in) circuit with AND, OR, and oracle gates. An oracle gate takes n inputs and produces m outputs. When the oracle is instantiated by a function $f\colon \{0,1\}^n \to \{0,1\}^m$ the computation proceeds as usual; the output of each oracle gate is f evaluated on its input. In this lecture we will use the oracle to encode the input to the circuit of interest, but in general oracles can be used to provide other capabilities, for example to run a "subroutine" f that the circuit may not be able to do on its own. The (worst-case) *query complexity* of an oracle circuit is the maximum number of oracle gates over all directed path in the underlying graph.

For simplicity today we will talk about circuits, but oracles can also be provided to Turing Machines. Such machines have an additional oracle tape and a special oracle state. When the machine goes into the oracle state, the oracle (which is a function f of type $\{0,1\}^* \to \{0,1\}^*$) the value x of the oracle tape is replaced with $f(x)$.

One of the simplest problems in this setting is database search. Say we have a database of 2^n items and we want to find an item of a given type (if it exists). We can describe the possible items by strings in $\{0,1\}^n$ and their property (whether they are of the desired type or not) by a value $f(x) \in \{0,1\}$. The problem of database search is given $f\colon \{0,1\}^n \to \{0,1\}$, find an $x \in \{0,1\}^n$ such that $f(x) = 1$ if it exists. The corresponding decision problem is to determine the existence of such an x.[1] This amounts to computing the OR of all the values $f(x)$ as x ranges over $\{0,1\}^n$, so in particular the decision problem can be solved by an oracle circuit of size 2^n, which is equal to the "size" of f. Using a bit more work, we can design a circuit of size $\text{poly}(n) \cdot 2^n$ for the search version. Is it possible to do better?

Let us argue that any *deterministic* oracle circuit $C^?$ for the OR function (the decision version of database search) has query complexity, and therefore circuit size, at least 2^n. If the circuit made any fewer queries, it could be that f takes value zero on all of them and the value of $\text{OR}_x\ f(x)$ as x ranges over $\{0,1\}^n$ cannot be determined, so $C^?$ cannot compute the OR function.

It is no coincidence that this argument closely resembles some of the decision tree analyses we

[1] These are not NP problems: We do not require that solutions can be verified efficiently.

did in Lecture 1. Every boolean-valued oracle circuit $C^?$ with oracle type $\{0,1\}^n \to \{0,1\}$ can be simulated by a decision tree over the 2^n variables $f(x)$. The root variable of the decision tree is the first oracle entry queried by f, the variables at depth 1 correspond to the second query (depending on the answer to the first one), and so on. The depth of this decision tree equals the query complexity of f. In particular, since OR of 2^n variables requires decision tree depth 2^n, a circuit that computes the OR of its oracle values must make 2^n queries.

Randomized sublinear time A randomized circuit takes, in addition to its regular inputs, some randomness $r \in \{0,1\}^m$. This is initialized with a sequence of uniformly random bits and so the output of such a circuit is a random variable. We say that the circuit computes g with error at most ε if for all x, the probability that the circuit on input x does not output $g(x)$ is at most ε. The definition easily extends to oracle circuits. In general, randomness does not help much in database search:

Claim 1. *If C^f computes $\mathrm{OR}_x\ f(x)$ with error at most $1/4$ for all $f\colon \{0,1\}^n \to \{0,1\}$ then $C^?$ makes at least 2^{n-1} queries.*

Proof. Assume $C^?$ makes at most q queries. For every fixing of the randomness r there is a decision tree T such that $T(f) = C^f(r)$, so there is a distribution T over decision trees of depth at most q such that the random variables $T(f)$ and C^f are identical.

Now consider the following distribution f on functions $f\colon \{0,1\}^n \to \{0,1\}$: $f(x)$ is set to one at a uniformly random input x and zero everywhere else. We claim that for any decision tree T of depth at most q,

$$\Pr_f[T(f) \neq T(0)] \leq \frac{q}{N} \tag{1}$$

where 0 is the all zero function and $N = 2^n$. Indeed, if $x_1, \ldots, x_q$ are the queries of q, then probability that $f(x_i)$ is zero conditioned on $f(x_1) = \cdots = f(x_{i-1}) = 0$ is exactly $(N-i)/(N-i-1)$ as long as the queries are all distinct, and even larger otherwise. Therefore $T(f)$ equals $T(0)$ with probability at least $(N-q)/N$. By linearity, (1) must also hold for every distribution on decision trees T, so in particular

$$\Pr_{f,r}[C^f(r) \neq C^0(r)] = \Pr_{f,T}[T(f) \neq T(0)] \leq \frac{q}{N}.$$

So if $C^?$ computes f with error at most $1/4$ by a union bound we must have

$$\begin{aligned} 1 &= \Pr_f[\mathrm{OR}_x f(x) \neq 0] \\ &\leq \Pr_{f,r}[C^f(r) \neq \mathrm{OR}_x f(x)] + \Pr_r[C^0(r) \neq 0] + \Pr_{f,r}[C^f(r) \neq C^0(r)] \\ &= \frac{1}{4} + \frac{1}{4} + \frac{q}{N} \end{aligned}$$

from where $q \geq N/2 = 2^{n-1}$. □

From the proof we see the reason why randomness does not help reduce the query complexity of database search: The database may have only one marked item. Even with randomness it takes a linear number of queries (in the database size) to locate this item. The situation changes dramatically if a large fraction of items is marked.

A *promise decision problem* is a pair of disjoint subsets (YES, NO) of the set of possible instances. Unlike in our definition of (total) decision problems, the subsets need not form a partition. We say an algorithm (or circuit) solves the promise problem if it outputs 1 on yes instances and 0 or no instances; it is allowed to output anything it wants on the other instances. A *promise search problem*

is a search problem together with a subset *YES* of instances among those that have a solution. An algorithm solves the promise search problem if it finds a solution for all the yes instances.

Now consider the promise problem of finding an x such that $f(x) = 1$, where the set of "yes" instances consists of those f such that $f(x) = 1$ for at least $\delta \cdot 2^n$ items x. This problem can be solved with error say $1/4$ by a randomized oracle circuit of size $\text{poly}(n)/\delta$. (In the midterm exam you will show that this is close to optimal in the sense that $\Omega(1/\delta)$ queries are necessary.) The circuit queries f at $2/\varepsilon$ random x and outputs the first x such that $f(x) = 1$. Assuming f is a "yes" instance, the probability that a solution x is missed is at most $(1-\delta)^{3/\delta} \leq e^{-2} < 1/4$. In contrast, any deterministic oracle circuit must make at least $(1-\delta)2^n$ queries (and so has to be at least this large) by a usual decision tree argument.

To conclude, we have an example of a promise problem which has randomized circuits of size polylogarithmic in its input, but requires deterministic circuits of at least linear size. In particular, the gap between the query complexity of randomized and deterministic circuits for this problem is exponential. The role of the promise is crucial here: For total (non-promise) problems, this gap can be at most quadratic.

2 Classical and quantum register machines

Quantum computation is an extension of randomized computation that captures the ability of quantum mechanical devices to occupy a "superposition" of states. We begin by describing a "new" model of classical randomized register machine, argue that it is equivalent to a randomized circuit, and then equip this model with quantum abilities.

A *randomized register machine* consists of a finite number of registers and a sequence of instructions. There are two types of instructions:

- $\text{T}(i,j,k)$ which replaces the contents of register k with c XOR (a AND b), where a, b, c are the respective contents of registers i, j, k, and
- $\text{R}(i)$ which replaces the contents of register i with a uniformly random bit.

An *oracle register machine* (for oracle type $\{0,1\}^n \to \{0,1\}^m$) has an additional instruction $\text{E}(i,j) = \text{E}(i_1, \ldots i_n, j_1, \ldots, j_m)$ which first reads x as the contents of registers $i = i_1 \ldots i_n$ then XORs the value of registers $j = j_1 \ldots j_m$ with the value obtained by evaluating the oracle at x.

The input to the register machine (if any) is given in the leftmost registers, while the rest are initialized to zero. The machine then executes the instructions in sequence. After all instructions are executed the output is read from the leftmost registers.

It is not difficult to see that a circuit whose input length, randomness, and circuit size are at most s can be simulated by a machine with $O(s)$. In the forward direction, there is one register for every input, every gate, and every random bit. The registers corresponding to the random bits are initialized with R and each gate of the circuit is emulated by a suitable short sequence of T instructions. In the other direction, a machine with m registers and s instructions can be simulated by a circuit of size $O(ms)$ along the lines of our simulation of Turing Machines by circuits.

Once the input and oracle are fixed, the state of an n-register machine at any point in time is a probability distribution p over all 2^n possible register values $x \in \{0,1\}^n$. Every instruction induces a transformation of this distribution p into a new distribution p': For example the instruction $\text{R}(1)$

randomizes the value of register 1 while leaving all others intact, in which case

$$p'(0y) = p'(1y) = \frac{1}{2}p(0y) + \frac{1}{2}p(1y) \quad \text{for all } y \in \{0,1\}^{n-1}.$$

Similarly, the instruction $\mathrm{T}(1,2,3)$ induces the transformation

$$p'(abc'y) = p(abcy), \quad \text{where } c' = c \text{ XOR } (a \text{ AND } b).$$

The transformations from p to p' are *linear* and *stochastic*: Each $p'(x')$ is a nonnegative linear combination of $p(x)$s, and the contributions of each $p(x)$ add up to one. The composition of stochastic linear transformations is also a stochastic linear transformation, so the final state p is also a probability distribution.

Quantum register machines In a quantum system with n "qubits", each of the 2^n possible register values x corresponds to a vector $|x\rangle$ in 2^n-dimensional Hilbert space. The vectors $|x\rangle$ form an orthonormal basis: They are orthogonal vectors of unit length. The state of the quantum system can be any unit vector in 2^n dimensions, namely any vector of the form

$$v = \sum_{x\in\{0,1\}^n} a(x) \cdot |x\rangle, \quad \text{where} \sum_{x\in\{0,1\}^n} a(x)^2 = 1.$$

The coefficients $a(x)$ may be positive, zero, or negative. The state of the quantum system can only be manipulated via orthogonal transformations, namely linear transformations that preserve the length of vectors.[2]

One important difference between stochastic and orthogonal transformations is that the latter are invertible, while the former are in general not. For this reason, any quantum process, including a computation, can always be reversed: The input can be "uncomputed" from the output. This prevents a quantum system from performing simple operations like erasing its memory. However, irreversible operations can be simulated by reversible ones. For example, if the memory of the quantum computer consists of two parts x and y and the x part needs to be erased, its contents can simply be moved to the y part via the orthogonal transformation that sends quantum state $\sum a(x,y)|x,y\rangle$ to quantum state $\sum a(x,y)|0, x \text{ XOR } y\rangle$. This transformation is a permutation of the basis vectors, so it is orthogonal. With this in mind, we can define the quantum variant of the register machine.

A *quantum oracle register machine* has the same syntax as a classical register machine, except that we replace the instruction $\mathrm{R}(i)$ by a new instruction $\mathrm{H}(i)$. The register of the machine is a quantum system with a finite number of qubits. Each quantum instruction specifies an orthogonal transformation on the quantum state of the register that we will now describe. To do this, it is sufficient to say how the transformation acts on the basis vectors, as it can be extended to all other vectors by linearity.

- A Toffoli instruction $\mathrm{T}(i,j,k)$ maps vector $|abcy\rangle$ to $|abc'y\rangle$, where $c' = c$ XOR $(a$ AND $b)$, and $abcy$ indicates the string with a in position i, b in position j, c in position k, and y in the other positions.
- A Hadamard instruction $\mathrm{H}(i)$ maps vector $|0y\rangle$ to $\frac{1}{\sqrt{2}}(|0y\rangle + |1y\rangle)$ and $|1y\rangle$ to $\frac{1}{\sqrt{2}}(|0y\rangle - |1y\rangle)$, where the fixed entry is in position i.

[2] In a general quantum system they can be complex numbers, in which case the sum of squares of the *amplitudes* must equal one, and unitary transformations are allowed to move between states.

An oracle instruction $\mathrm{E}(i,j)$ maps vector $|xyz\rangle$ to $|xy'z\rangle$, where x is the value of the registers indexed by i, y is the value of the registers indexed by j, and $y' = y \text{ XOR } f(x)$.

It needs to be verified that these transformations are indeed orthogonal. Toffoli and oracle instructions map basis vectors to other basis vectors, so lengths are preserved as long as the map between these vectors is invertible (namely a permutation). This follows from the fact that applying the same gate twice results in restoring the original state. Geometrically, the Hadamard gate represents a rotation by 45 degrees followed by a reflection across the $|0\rangle$-axis of each pair of vectors $|0y\rangle, |1y\rangle$, which also preserves vector length.

At the end of the computation, machine is in some quantum state $v = \sum a(x) \cdot |x\rangle$. To observe the register a *measurement* is performed: The resulting value is x with probability $a(x)^2$.

The Toffoli and Hadamard instructions are universal in the sense that they can be combined to implement any orthogonal transformation in which the amplitudes are square roots of inverse powers of two.

The power of quantum computation Quantum register machines can simulate classical ones. The invariant maintained by the simulation is that if the classical machine is in state x with probability $p(x)$ then the quantum machine will be in quantum state $\sum \sqrt{p(x)} \cdot |x\rangle$ with $a(x)$. This is true initially, and by design for the T and E instructions, as they induce the same permutation on the probability and amplitude vectors. A classical instruction $\mathrm{R}(i)$ can be simulated by the quantum instruction $\mathrm{H}(i)$ provided it is applied to a previously unused register. The resulting quantum change of state is

$$\sum a'(yb) \cdot |yb\rangle = \sum a(y0) \cdot \frac{|y0\rangle + |y1\rangle}{\sqrt{2}}$$

where the last bit refers to register i, and therefore

$$\sqrt{a'(y0)} = \sqrt{a'(y1)} = \sqrt{a(y0)/2}$$

which matches the change in probabilities of the classical machine, provided the i-th register was initially a zero with probability 1.

In this simulation all of the amplitudes of the quantum machines are positive. One property of quantum computation that makes it potentially more powerful than randomized classical computation are the negative amplitudes: While probabilities can only add up, amplitudes can also cancel out.

3 Quantum database search

We now consider the search problem of finding a marked item in the database, provided that exactly one such item exists. Formally, given $f\colon \{0,1\}^n \to \{0,1\}$, the task is to find x such that $f(x) = 1$ under the promise that exactly one such x exists. By the proof of Claim 1 any randomized circuit for this search problem must make $\Omega(2^n)$ queries. In contrast,

Theorem 2. *There is a quantum circuit of size* $\mathrm{poly}(n) \cdot 2^{n/2}$ *that, given oracle access to* $f\colon \{0,1\}^n \to \{0,1\}$ *finds* x *such that* $f(x) = 1$ *with probability* $1 - O(2^{-n/2})$*, provided that a unique such* a *exists.*

The uniqueness restriction can be removed using our reduction from general search to search with a unique solution from Lecture 6: If the number of x such that $f(x) = 1$ is between 2^{k-2} and 2^{k-1},

then the circuit can filter out those solutions such that $h(x) \neq 0$ where $h\colon \{0,1\}^n \to \{0,1\}^k$ is a random hash function. By Lemma 7 from Lecture 6, the remaining solution is unique with constant probability.

The quantum algorithm behind Theorem 2 is best described geometrically. The algorithm maintains a quantum state v that is initially "far" from the solution $|x\rangle$ in the sense that the two vectors are almost orthogonal. Then a sequence of steps is performed, where in each step the angle between v and $|x\rangle$ drops by about $2^{-n/2}$ radians. After about $2^{n/2}$ steps the inner product between the two becomes close to one and a measurement yields the outcome x with high probability.

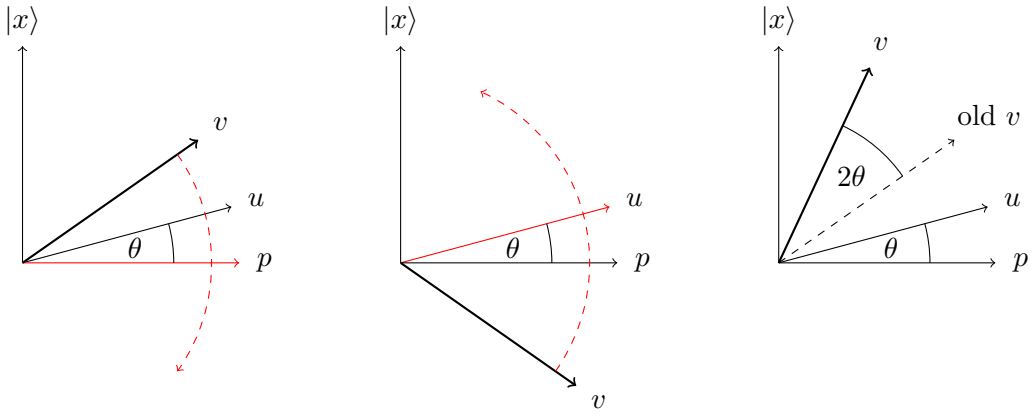
All the action takes place in the plane spanned by the vectors $|x\rangle$, u and p, where

$$u = \frac{1}{2^{n/2}} \sum_{y \in \{0,1\}^n} |y\rangle \quad \text{and} \quad p = \frac{1}{\sqrt{2^n - 1}} \sum_{y \neq x} |y\rangle.$$

Since u is a linear combination of p and $|x\rangle$, these three indeed lie in the same plane. Let θ be the angle between u and p. Then the dot product of $|x\rangle$ and u equals the cosine of $\pi/2 - \theta$, which is the sine of θ, so

$$\sin\theta = |x\rangle \cdot u = 2^{-n/2}$$

from where by Taylor expansion $\theta = 2^{-n/2} - O(2^{-3n/2})$. Now we can rotate u by an angle of 2θ towards $|x\rangle$ by reflecting it twice, first about p, then about u:



After t steps the angle between v and p equals $(2t+1)\theta$, so if we choose $2t+1$ to equal the closest integer to $\pi/2\theta$, the angle between v and $|x\rangle$ becomes at most θ so the square of their inner product, which equals the probability that the measurement outputs $|x\rangle$, equals $1 - O(\theta) = 1 - O(2^{-n/2})$.

To summarize, here is a high-level description of the circuit:

> Create the state $v = 2^{-n/2} \sum_{y \in \{0,1\}^n} |y\rangle$ in the first n registers.
> Repeat for t times:
> Reflect v about u
> Reflect v about p
> Measure and output the content of the first n registers.

It remains to show how the vectors and rotations can be implemented by a quantum circuit. The vector u is obtained by applying the H instruction on n independent qubits initialized to zero. The composed transformation is called the Hadamard transform on n bits.

A reflection about a vector w is a transformation that sends w to itself and any vector w' orthogonal to w to $-w'$. If w is a basis vector, say $w = |0^n\rangle$, this can be implemented as follows: First, compute

the predicate $y \neq 0^n$ into a fresh register. Then apply the transformation $|y0\rangle \to |y0\rangle$, $|y1\rangle \to -|y1\rangle$ where the last bit refers to the fresh register. The combined transformation sends $|0^n 0\rangle$ to itself and $|y0\rangle$ to $-|y1\rangle$ for every $y \neq 0$. Finally, compute $y \neq 0^n$ into the fresh register again. This restores the value of the fresh register to zero.

To reflect about u, we can first apply any transformation T that maps u to the basis vector $|0^n\rangle$, then apply a rotation around $|0^n\rangle$, and finally apply the inverse of T. T can be chosen as the Hadamard transform, which is its own inverse.

Finally, we show how to reflect v about p. For this it is sufficient to reflect v about the plane perpendicular to $|x\rangle$. This is the transformation that sends $|x\rangle$ to $-|x\rangle$ and $|y\rangle$ to itself for every $y \neq x$.

To implement this reflection, we first evaluate f on v and store the result into a fresh register i. Recall that if v is the basis vector $|y0\rangle$, the outcome of the evaluation is $|y1\rangle$ if $y = x$ and $|y0\rangle$ otherwise, where the last bit indicates the contents of the fresh register. We then send $|y0\rangle$ to itself and $|y1\rangle$ to $-|y1\rangle$. Composing the two transformations has the effect of sending $|x0\rangle$ to $-|x1\rangle$ and $|y0\rangle$ to itself for every other y. If we apply the evaluation gate again, $|x0\rangle$ gets mapped to $-|x0\rangle$ while $|y0\rangle$ remains invariant. This is exactly the desired reflection.

Each rotation of v by 2θ uses a number of instructions polynomial in n, so the overall number of instructions is $\text{poly}(n) \cdot 2^{n/2}$ as desired.

4 Quantum query complexity

We now sketch the proof that Theorem 2 is close to optimal: Any quantum register machine for searching a database on 2^n bits that succeeds with constant probability must make $\Omega(2^{n/2})$ queries to the database.

Theorem 3. *The probability of any outcome of a quantum register machine that makes at most q queries to oracle $f\colon \{0,1\}^n \to \{0,1\}$ is a polynomial of degree at most $2q$ in the 2^n variables $f(x)$, $x \in \{0,1\}^n$.*

Proof. We argue that the amplitudes of the (quantum) state of the register are polynomials of degree i after the i-th query, Initially the state does not depend on f, so each amplitude is a constant. Toffoli and Hadamard instructions are linear transformations of the amplitudes, so the degree of the polynomials is not affected by them. The change in amplitude by oracle instructions can be described by the equation

$$a'(xyz) = (1 - f(x))a(xyz) + f(x)a(x\overline{y}z)$$

which increases the degree by at most one. After all q queries, each amplitude is a degree q polynomial. Since a measurement is the square of an amplitude, it is a polynomial of degree at most $2q$ in the $f(x)$s. □

If a q query quantum machine finds unique assignments to f with error ε, then such a machine can solve the promise decision problem whose yes instances are those f that evaluate to one at exactly one input and whose no instance is the all zero function with the same error. By Theorem 3, there must then exist a polynomial p of degree $2q$ in $N = 2^n$ variables $z_1, \ldots, z_N$ such that $p(0^N) \leq \varepsilon$, $p(z) \geq 1 - \varepsilon$ for any $z \in \{0,1\}^N$ that has exactly one 1-entry, and $p(z) \in [0,1]$ for all $z \in \{0,1\}^N$. Using some approximation theory, it is possible to show that such a polynomial requires degree $\Omega(\sqrt{N \log(1/\varepsilon)})$. It follows that the quantum register machine must make $\Omega(2^{n/2}\sqrt{\log(1/\varepsilon)})$ queries.

References

Relations between deterministic and randomized query complexity and the degree of polynomials representing a given function were first studied by Nisan and Szegedy. They also show the degree lower bound stated in Section 4. Theorem 3 was observed by Beals, Buhrman, Cleve, Mosca, and De Wolf. Our presentation of quantum computation follows the Arora-Barak textbook which gives more detail and additional references.