Introduction to Quantum Computing

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU Math 500

Subsection 1

Computing with Superpositions

Introducing Quantum Parallelism

- Many quantum algorithms use quantum analogs of classical computation as at least part of their computation.
- Quantum algorithms often start by:
  - Creating a quantum superposition;
  - Then feeding it into a quantum version U_f of a classical circuit that computes a function f.
- This setup is called **quantum parallelism**.
- It accomplishes nothing by itself - any algorithm that stopped at this point would have no advantage over a classical algorithm.
- However, the construction leaves the system in a state that quantum algorithm designers have found a useful starting point.

The Walsh-Hadamard Transformation

- Quantum parallelism starts by using the Walsh-Hadamard transformation.
- This is a generalization of the Hadamard transformation that creates a superposition of all input values.
- Recall that the Hadamard transformation H applied to $|0\rangle$ creates a superposition state $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$.
- Applied to n qubits individually, all in state $|0\rangle$, H generates a superposition of all 2^n standard basis vectors, which can be viewed as the binary representation of the numbers from 0 to $2^n - 1$:

$$
\begin{aligned}
&(H \otimes H \otimes \cdots \otimes H)|00\ldots0\rangle \\
&= \frac{1}{\sqrt{2^n}}((|0\rangle + |1\rangle) \otimes (|0\rangle + |1\rangle) \otimes \cdots \otimes (|0\rangle + |1\rangle)) \\
&= \frac{1}{\sqrt{2^n}}(|0\ldots00\rangle + |0\ldots01\rangle + |0\ldots10\rangle + \cdots + |1\ldots11\rangle) \\
&= \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle.
\end{aligned}
$$

The Walsh-Hadamard Transformation (Cont'd)

- The **Walsh**, or **Walsh-Hadamard**, transformation

$$W = H \otimes H \otimes \cdots \otimes H$$

applies H to each of the qubits in an n-qubit state.

- Using $N = 2^n$, we may write

$$W|0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle.$$

- In the standard basis, the matrix for the n-qubit Walsh-Hadamard transformation is a $2^n \times 2^n$ matrix W, with entries W_{rs}, such that

$$W_{sr} = W_{rs} = \frac{1}{\sqrt{2^n}}(-1)^{r \cdot s}, \quad 0 \leq r, s \leq 2^n - 1,$$

where $r \cdot s$ is the number of common one-bits in s and r.

The Walsh-Hadamard Transformation (Cont'd)

- To see this equality, note that

$$W(|r\rangle) = \sum_s W_{rs}|s\rangle.$$

- Let $r_{n-1}\dots r_0$ be the binary representation of r.
- Let $s_{n-1}\dots s_0$ be the binary representation of s.
- Then we have

$$\begin{aligned} W(|r\rangle) &= (H\otimes\cdots\otimes H)(|r_{n-1}\rangle\otimes\cdots\otimes|r_0\rangle) \\ &= \frac{1}{\sqrt{2^n}}(|0\rangle+(-1)^{r_{n-1}}|1\rangle)\otimes\cdots\otimes(|0\rangle+(-1)^{r_0}|1\rangle) \\ &= \frac{1}{\sqrt{2^n}}\sum_{s=0}^{2^n-1}(-1)^{s_{n-1}r_{n-1}}|s_{n-1}\rangle\otimes\cdots\otimes(-1)^{s_0 r_0}|s_0\rangle \\ &= \frac{1}{\sqrt{2^n}}\sum_{s=0}^{2^n-1}(-1)^{s\cdot r}|s\rangle. \end{aligned}$$

Quantum Parallelism

- Any transformation of the form $U_f = |x, y\rangle \rightarrow |x, y \oplus f(x)\rangle$ is linear.
- Therefore, it acts on a superposition $\sum a_x|x\rangle$ of input values,

$$U_f : \sum_x a_x|x, 0\rangle \rightarrow \sum_x a_x|x, f(x)\rangle.$$

- Consider the effect of applying U_f to the superposition of values from 0 to $2^n - 1$ obtained from the Walsh transformation:

$$U_f : (W|0\rangle) \otimes |0\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|0\rangle \rightarrow \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle|f(x)\rangle.$$

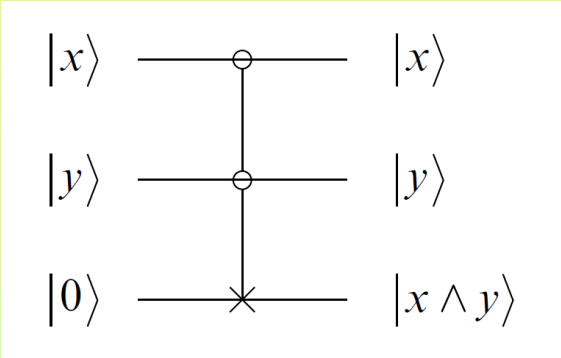
- After only one application of U_f, the superposition now contains all of the 2^n function values $f(x)$ entangled with their corresponding input value x.
- This effect is called **quantum parallelism**.

Remarks

- n qubits enable us to work simultaneously with 2^n values.
- We have the ability to hold exponentially many computed values in a linear amount of physical space.
- So quantum parallelism, in some sense, circumvents the time/space trade-off of classical parallelism.
- However, this effect is less powerful than it may initially appear.
- First, it is possible to gain only limited information.
- These 2^n values of f are not independently accessible.
- We can gain information only from measuring the states.
- But measuring in the standard basis will project the final state onto a single input /output pair $|x, f(x)\rangle$, and a random one at that.

Example

- We use the basic setup of quantum parallelism.
- We illustrate how useless the raw superposition arising from quantum parallelism is on its own, without any additional transformations.
- The controlled-controlled-NOT (Toffoli) gate, T, computes the conjunction of two values.
- Take as input the superposition of all possible bit combinations of x and y together with a single-qubit register, initially set to $|0\rangle$, that is to contain the output.



- We use quantum parallelism to construct this input state in the standard way:

$$\begin{aligned} W(|00\rangle) \otimes |0\rangle &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) \otimes |0\rangle \\ &= \frac{1}{2}(|000\rangle + |010\rangle + |100\rangle + |110\rangle). \end{aligned}$$

Example (Cont'd)

- Applying the Toffoli gate T to this superposition of inputs yields

$$T(W|00\rangle \otimes |0\rangle) = \frac{1}{2}(|000\rangle + |010\rangle + |100\rangle + |111\rangle).$$

- This superposition can be viewed as a truth table for conjunction.
- The values of x, y and $x \wedge y$ are entangled in such a way that measuring in the standard basis will give one line of the truth table.
- Computing the AND using quantum parallelism, and then measuring in the standard basis, gives no advantage over classical parallelism.
  - Only one result is obtained;
  - We cannot even choose which result we get.

Subsection 2

Notions of Complexity

Circuit Families and Complexity

- A circuit family $\mathcal{C} = \{C_n\}$ consists of circuits C_n indexed by the maximum input size for that circuit.
- That is, the circuit C_n handles input of size n (bits or qubits).
- The **complexity** of a circuit C is defined to be the number of simple gates in the circuit, where the set of simple gates under consideration must be specified.
- Any of the finite sets of gates discussed in a previous section may be used.
- Alternatively, the infinite set consisting of all single qubit operations together with the C_{not} may be used.

Circuit Complexity or Time Complexity

- The **circuit complexity**, or **time complexity**, of a family of circuits $\mathcal{C} = \{C_n\}$ is the asymptotic number of simple gates in the circuits, expressed as a function of the input size.
- The circuit complexity for a circuit family $\mathcal{C} = \{C_n\}$ is $O(f(n))$ if the size of the circuit is bounded by $O(f(n))$
- This means that the function $t(n) = |C_n|$ satisfies

$$t(n) \in O(f(n)).$$

- Any of the simple gate sets mentioned earlier give the same asymptotic circuit complexity.

Uniformity

- Circuit complexity models are nonuniform in that different, larger circuits are required to handle larger input sizes.
- Both quantum and classical Turing machines, by contrast, propose a single machine that can handle arbitrarily large input.
- The nonuniformity of circuit models makes circuit complexity more complicated to define than Turing machine models.
- Here, in contrast to uniform models, complexity can be *hidden* in the complexity of constructing the circuits C_n themselves, even if the size of the circuits C_n is asymptotically bounded.
- To get sensible notions of complexity, in particular to obtain circuit complexity measures similar to Turing machine based ones, a separate *uniformity condition* must be imposed.
- Both quantum and classical circuit complexity use similar uniformity conditions.

Consistency

- In addition to uniformity, a requirement that the behavior of the circuits C_n in a circuit family $\mathcal{C}$ behave in a consistent manner is usually imposed as well.
- This *consistency condition* is usually phrased in terms of a function $g(x)$, and says that all circuits $C_n \in \mathcal{C}$ that can take x as input give $g(x)$ as output.
- This condition is sometimes misunderstood to include restrictions on the sorts of functions $g(x)$ a consistent circuit family can compute.
- For this reason, and to generalize easily to the quantum case, we phrase this same consistency condition without explicit reference to a function $g(x)$.

Consistency and Uniformity Conditions

Consistency Condition: A quantum or classical circuit family $\mathcal{C}$ is **consistent** if its circuits C_n give consistent results:

> For all $m < n$, applying circuit C_n to input x of size m must give the same result as applying C_m to that input.

- The most common uniformity condition, and the one we impose here, is the polynomial uniformity condition.

Uniformity Condition: A quantum or classical circuit family $\mathcal{C} = \{C_n\}$ is **polynomially uniform** if there exists a polynomial-time classical algorithm that generates the circuits.

- In other words, $\mathcal{C}$ is polynomially uniform if there exists a polynomial $f(n)$ and a classical program that, given n, constructs the circuit C_n in at most $O(f(n))$ steps.

Uniformity and Consistency in the Classical Case

- The relation between the circuit complexity of polynomially uniform, consistent circuit families and the Turing machine complexity is understood for both the classical and quantum case.
- In the classical case, for any classical function $g(x)$ computable on a Turing machine in time $O(f(n))$, there is a polynomially uniform, consistent classical circuit family that computes $g(x)$ in time $O(f(n)\log f(n))$.
- Conversely, a polynomially uniform, consistent family of Boolean circuits can be simulated efficiently by a Turing machine.

Uniformity and Consistency in the Quantum Case

- In the quantum case, Yao has shown that any polynomial time computation on a quantum Turing machine can be computed by a polynomially uniform, consistent family of polynomially sized quantum circuits.
- As in the classical case, demonstrating that any polynomially uniform, consistent family of quantum circuit can be simulated by a quantum Turing machine is straightforward.
- Here, we are not concerned with sublinear complexity differences, i.e., asymptotic differences of at most a polynomial in $\log(f(n))$.
- So we discuss quantum complexity in terms of circuit complexity, with the polynomial uniformity condition, instead of using quantum Turing machines.

Black Boxes or Oracles

- The earliest quantum algorithms solve **black box**, or **oracle**, problems.
- A **classical black box** outputs $f(x)$ upon input of x.
- A **quantum black box** behaves like U_f, outputting $\sum_x \alpha_x |x, f(x) \oplus y\rangle$ upon input of $\sum_x \alpha_x |x\rangle|y\rangle$.
- Black boxes are purely theoretical constructs.
- A black box may or may not have an efficient implementation.
- For this reason, they are often called **oracles**.
- The black box terminology (unviewable interior) emphasizes that only the output of a black box can be used to solve the problem, not anything about its implementation or any of the intermediate values computed along the way.

Query Complexity

- The most common type of complexity discussed with respect to black box problems is **query complexity**:

  The number of calls to the oracle required to solve the problem.

- Black box algorithms of low query complexity, algorithms that solve a black box problem with few calls to the oracle, are only of practical use if the black box has an efficient implementation.
- The black box approach is very useful, however, in establishing **lower bounds** on the circuit complexity of a problem.
- If the query complexity is $\Omega(N)$ - in other words, at least $\Omega(N)$ calls to the oracle are required - then the circuit complexity must be at least $\Omega(N)$.

Relating Quantum with Classical Complexity

- Black boxes have been used to establish lower bounds on the circuit complexity for quantum algorithms.
- Their first use was to show that the quantum query complexity of certain black box problems was strictly less than the classical query complexity.
- The number of calls to a quantum oracle needed to solve certain problems is strictly less than the required number of calls to a classical oracle to solve the same problem.

Relating Quantum with Classical Complexity

- The first few genuinely quantum algorithms that we describe solve black box problems:
  - Deutsch's problem;
  - the Deutsch-Jozsa problem;
  - the Bernstein-Vazirani problem;
  - Simon's problem.
- Grover's Result: It takes only $O(\sqrt{N})$ calls to a quantum black box to solve an unstructured search problem over N elements, whereas the classical query complexity of unstructured search is $\Omega(N)$.

Subsection 3

A Simple Quantum Algorithm

Deutsch's Problem

Deutsch's Problem: Given a Boolean function $f : \mathbb{Z}_2 \to \mathbb{Z}_2$, determine whether f is constant.

- Deutsch's quantum algorithm requires only a single call to a black box for U_f to solve the problem.
- Any classical algorithm requires two calls to a classical black box for C_f, one for each input value.
- The key to Deutsch's algorithm is the nonclassical ability to place the second qubit of the input to the black box in a superposition.

Deutsch's Algorithm

- Recall that U_f for a single bit function f:
  - Takes two qubits of input;
  - Produces two qubits of output.
- On input $|x\rangle|y\rangle$, U_f produces

$$|x\rangle|f(x) \oplus y\rangle.$$

- So when $|y\rangle = |0\rangle$, the result of applying U_f is $|x\rangle|f(x)\rangle$.
- The algorithm applies U_f to the two-qubit state $|+\rangle|-\rangle$, where:
  - The first qubit is the superposition $|+\rangle = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$;
  - The second qubit is the superposition $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$.

Deutsch's Algorithm (Cont'd)

- We obtain

$$\begin{aligned} U_f(|+\rangle|-\rangle) &= U_f\left(\tfrac{1}{2}(|0\rangle+|1\rangle)(|0\rangle-|1\rangle)\right) \\ &= \tfrac{1}{2}\left(|0\rangle(|0\oplus f(0)\rangle-|1\oplus f(0)\rangle)\right. \\ &\quad \left.+|1\rangle(|0\oplus f(1)\rangle-|1\oplus f(1)\rangle)\right). \end{aligned}$$

- In other words,

$$U_f(|+\rangle|-\rangle) = \frac{1}{2}\sum_{x=0}^{1}|x\rangle(|0\oplus f(x)\rangle-|1\oplus f(x)\rangle).$$

Deutsch's Algorithm (Cont'd)

- Suppose $f(x) = 0$.

$$\frac{1}{\sqrt{2}}(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle) = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = |-\rangle.$$

- Suppose $f(x) = 1$.

$$\frac{1}{\sqrt{2}}(|0 \oplus f(x)\rangle - |1 \oplus f(x)\rangle) = \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -|-\rangle.$$

- Therefore,

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

- For f constant, $(-1)^{f(x)}$ is just a physically meaningless global phase, so the state is simply $|+\rangle|-\rangle$.
- For f not constant, the term $(-1)^{f(x)}$ negates exactly one of the terms in the superposition so, up to a global phase, the state is $|-\rangle|-\rangle$.

Deutsch's Algorithm (Cont'd)

- Now we apply the Hadamard transformation H to the first qubit.
- We finally measure it (in the default standard basis).
- Then, with certainty, we obtain:
  - $|0\rangle$, if f is constant;
  - $|1\rangle$, if f is not constant.
- Thus with a single call to U_f we can determine, with certainty, whether f is constant or not.

Comments

- It may be surprising that this algorithm succeeds with certainty.
- The most commonly remembered aspect of quantum mechanics is its probabilistic nature.
- This leads to the naive expectation that anything done with quantum means must be probabilistic.
- At least, it makes one think that anything that exhibits peculiarly quantum properties must be probabilistic.
- We already know, from our study of quantum analogs to classical computations, that the first of these expectations does not hold.
- The algorithm for Deutsch's problem shows that even inherently quantum processes do not have to be probabilistic.

Subsection 4

Quantum Subroutines

Importance of Uncomputing Qubits

- In quantum computation, uncomputing qubits used temporarily as part of subroutines is crucial even when conserving space and reusing qubits is not an issue.
- Failing to uncompute temporary qubits can result in entanglement between the computational qubits and the temporary qubits.
- This entanglement may destroy the calculation.
- If a subroutine claims to compute state $\sum_i \alpha_i |x_i\rangle$, it is not okay if it actually computes $\sum_i \alpha_i |x_i\rangle|y_i\rangle$ and throws away the qubits storing $|y_i\rangle$ unless there is no entanglement between the two registers.
- There is no entanglement if $\sum_i \alpha_i |x_i\rangle|y_i\rangle = (\sum_i \alpha_i |x_i\rangle) \otimes |y_i\rangle$, which can happen only if $|y_i\rangle = |y_j\rangle$, for all i and j.
- In general, the states $\sum_i \alpha_i |x_i\rangle$ and $\sum_i \alpha_i |x_i\rangle|y_i\rangle$ behave quite differently, even if we have access only to the first register of the second state.

Illustration Using Deutsch's Algorithm

- We illustrate the difference by showing how using the first state when expecting the second can mess up computation.
- Suppose we replace the black box U_f used in Deutsch's problem with the black box for V_f that outputs

$$V_f : |x, t, y\rangle \rightarrow |x, t \oplus x, y \oplus f(x)\rangle.$$

- We show that Deutsch's algorithm no longer works.

Illustration Using Deutsch's Algorithm (Cont'd)

- We begin with:
  - Qubit $|t\rangle$ in the state $|0\rangle$;
  - The first qubit, as before, in the state $|+\rangle$;
  - The third, as before, in the state $|-\rangle$.
- Apply V_f to obtain

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

- The first qubit is now entangled with the second qubit.

Illustration Using Deutsch's Algorithm (Cont'd)

- Because of this entanglement, applying H to the first qubit and then measuring it no longer has the desired effect.
- For example, suppose f is constant.
- Then the state is $(|00\rangle + |11\rangle)|-\rangle$.
- Applying $H \otimes I \otimes I$ results in the state

$$\frac{1}{2}(|00\rangle + |10\rangle + |01\rangle - |11\rangle)|-\rangle.$$

- The second and fourth terms do not cancel now.
- There is an equal chance of measuring the first qubit as $|0\rangle$ or $|1\rangle$.
- A similar calculation shows that, when the function is not constant, there is also an equal chance of measuring the first qubit as $|0\rangle$ or $|1\rangle$.
- Thus, we can no longer distinguish the two cases.
- Entanglement with $|t\rangle$ has destroyed the quantum computation.

A Remedy

- Had V_f properly uncomputed t so that at the end of the calculation it was in state $|0\rangle$, the algorithm would still work properly.
- For example, for f constant, we would have state

$$\frac{1}{2}(|00\rangle + |10\rangle + |00\rangle - |10\rangle)|-\rangle.$$

- The appropriate terms would cancel to yield $(|00\rangle)|-\rangle$.
- If a quantum subroutine claims to produce a state $|\psi\rangle$, it must not produce a state that looks like $|\psi\rangle$ but is entangled with other qubits.
- In particular, if a subroutine makes use of other qubits, by the end these qubits must not be entangled with the other qubits.
- The following quantum subroutines are careful to uncompute any auxiliary qubits so that at the end of the algorithm they are always in state $|0\rangle$.

Phase Change for a Subset of Basis Vectors

Aim: Change the phase of terms in a superposition $|\psi\rangle = \sum a_i |i\rangle$ depending on whether i is in a subset X of $\{0, 1, \ldots, N-1\}$ or not.

- More specifically, we wish to find an efficient implementation of the quantum transformation

$$S_X^{\phi} : \sum_{x=0}^{N-1} a_x |x\rangle \to \sum_{x \in X} a_x e^{\boldsymbol{i}\phi} |x\rangle + \sum_{x \notin X} a_x |x\rangle.$$

Comments on Efficiency

- We explained how to realize an arbitrary unitary transformation without regard to efficiency.
- Applying that algorithm blindly would give an implementation of S_X^ϕ using more than $N = 2^n$ simple gates.
- We now show how, for any efficiently computable subset X, the transformation S_X^ϕ can be implemented efficiently.
- An efficiently implementable S_X^ϕ is used in some of the quantum algorithms we describe later that outperform classical ones.

Role of U_f

- We can hope to implement S_X^ϕ efficiently only if there is an efficient algorithm for computing membership in X.
- More precisely, the Boolean function $f : \mathbb{Z}_{2^n} \to \mathbb{Z}_2$, where

$$f(x) = \begin{cases} 1, & \text{if } x \in X, \\ 0, & \text{otherwise} \end{cases}$$

  must be efficiently computable, say in polynomial time in n.
- Most subsets X do not have this property.
- For subsets X with this property, the main result of the preceding set implies that there is an efficient quantum circuit for U_f.
- Given such an implementation for U_f, we can compute S_X^ϕ using a few additional steps.

Implementation of the Phase Shift

- We proceed in steps:
  - Use U_f to compute f in a temporary qubit;
  - Use the value in that qubit to effect the phase change;
  - Uncompute f in order to remove any entanglement between the temporary qubit and the rest of the state.
- We encode as follows.

define $Phase_f(\phi)|x[k]\rangle =$

quibit $a[1]$	a temporary bit	(1)
$U_f\|x,a\rangle$	compute f in a	(2)
$K(\frac{\phi}{2})\|a\rangle$		(3)
$T(-\frac{\phi}{2})\|a\rangle$		(4)
$U_f^{-1}\|x,a\rangle$	uncompute f	(5)

Comments on the Implementation

- We know that
$$T\left(-\frac{\phi}{2}\right)K\left(\frac{\phi}{2}\right) = \begin{pmatrix} 1 & 0 \\ 0 & e^{\boldsymbol{i}\phi} \end{pmatrix}.$$
- So, together, Steps (3) and (4) shift the phase by $e^{\boldsymbol{i}\phi}$ if and only if bit a is one.
- Strictly speaking, we do not need to do Step (3) at all, since it is a physically meaningless global phase shift.
- But performing Step (3) makes it easier to see that we get the desired result.

Comments on the Implementation (Cont'd)

- Alternatively, we could replace Steps (3) and (4) by a single step

$$\bigwedge_1 K(\phi)|a\rangle|x_i\rangle,$$

  where i can be any of the qubits in register x.
- This is because placing a phase in any term of the tensor product is the same as placing it in any other term.
- We need to uncompute U_f in Step (5) to remove the entanglement between register $|x\rangle$ and the temporary qubit.
- This results in $|x\rangle$ being in the desired state, no longer entangled with the temporary qubits.

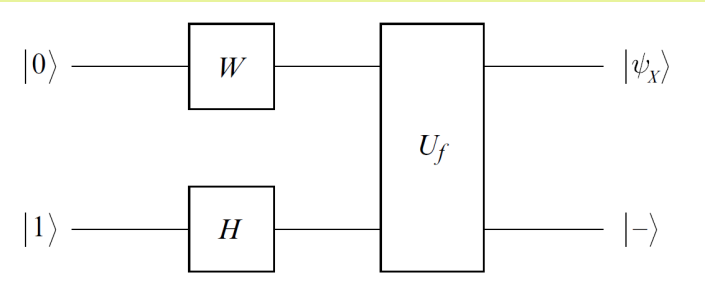
Special Case $\phi = \pi$

- The important special case $\phi = \pi$ has an alternative, surprisingly simple, implementation that generalizes the trick used in the algorithm for Deutsch's problem.
- Given U_f as above, the transformation S_X^π can be implemented by:
  - Initializing a temporary qubit b to $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$;
  - Using U_f to compute into this register.
- Consider $|\psi\rangle = \sum_{x \in X} a_x |x\rangle + \sum_{x \notin X} a_x |x\rangle$.
- We compute

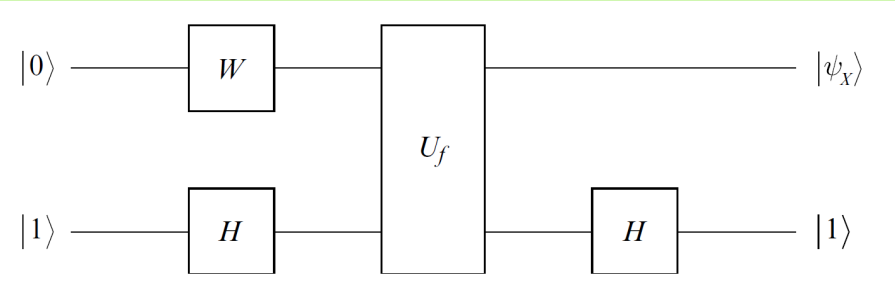
$$\begin{aligned} U_f(|\psi\rangle \otimes |-\rangle) &= U_f\left(\sum_{x \in X} a_x |x\rangle \otimes |-\rangle\right) + U_f\left(\sum_{x \notin X} a_x |x\rangle \otimes |-\rangle\right) \\ &= -\left(\sum_{x \in X} a_x |x\rangle \otimes |-\rangle\right) + \left(\sum_{x \notin X} a_x |x\rangle \otimes |-\rangle\right) \\ &= (S_X^\pi |\psi\rangle) \otimes |-\rangle. \end{aligned}$$

Special Case $\phi = \pi$ (Illustration)

- In particular, the following circuit, acting on the n-qubit state $|0\rangle$ together with an ancilla qubit in state $|1\rangle$ creates the superposition $|\psi_X\rangle = \sum(-1)^{f(x)}|x\rangle$:



- For elegance, and to be able to reuse the ancilla qubit, we may want to apply a final Hadamard transformation to the ancilla qubit, in which case the circuit is



Special Case $\phi = \pi$ (Geometry)

- Geometrically, when acting on the N-dimensional vector space associated with the quantum system, the transformation S_X^π is a reflection about the $(N-k)$-dimensional hyperplane perpendicular to the k-dimensional hyperplane spanned by $\{|x\rangle : x \in X\}$.
- A reflection in a hyperplane sends any vector $|v\rangle$ perpendicular to the hyperplane to its negative $-|v\rangle$.
- For any unitary transformation U, the transformation $US_X^\pi U^{-1}$ is a reflection in the hyperplane perpendicular to the hyperplane spanned by the vectors $\{U|x\rangle : x \in X\}$.

Special Case $\phi = \pi$ (Cont'd)

- We can write the result of applying S_X^π to the superposition $W|0\rangle$ as

$$\frac{1}{\sqrt{N}} \sum (-1)^{f(x)} |x\rangle,$$

where f is the Boolean function for membership in X,

$$f(x) = \begin{cases} 1, & \text{if } x \in X, \\ 0, & \text{otherwise.} \end{cases}$$

- Conversely, given a Boolean function f, we define

$$S_f^\pi := S_X^\pi,$$

where $X = \{x : f(x) = 1\}$.

State-Dependent Phase Shifts

Aim: Efficiently approximate to accuracy s the transformation on n-qubits that changes the phase of the basis elements by

$$|x\rangle \to e^{\boldsymbol{i}\phi(x)}|x\rangle,$$

where:

- The function $\phi(x)$ that describes the desired phase shift angle ϕ for each term x has an associated function $f : \mathbb{Z}_n \to \mathbb{Z}_s$ that is efficiently computable;
- The value of the i-th bit of $f(x)$ is the i-th term in the following binary expansion for $\phi(x)$:

$$\phi(x) \approx 2\pi \frac{f(x)}{2^s}.$$

State-Dependent Phase Shifts (Cont'd)

- The implementation can be only as efficient as the function f.
- Given a quantum circuit that efficiently implements U_f, we can perform the state-dependent phase shift in $O(s)$ steps in addition to 2 uses of U_f.
- The ability to compute f efficiently is a strong one, in that most functions do not have this property.

Changing Phase Subroutine

- We show how to implement the subprogram that changes the phase of an s-qubit standard basis state $|x\rangle$ by the angle $\phi(x) = \frac{2\pi x}{2^s}$.
- Let $P(\phi)$ be the transformation that shifts the phase in a qubit, if that bit is 1, but does nothing, if that bit is 0,

$$P(\phi) = T\left(-\frac{\phi}{2}\right) K\left(\frac{\phi}{2}\right) = \begin{pmatrix} 1 & 0 \\ 0 & e^{\boldsymbol{i}\phi} \end{pmatrix}.$$

- The following program performs the s-qubit transformation $Phase : |a\rangle \rightarrow \exp\left(\boldsymbol{i} 2\pi \frac{a}{2^s}\right)|a\rangle$:

define $Phase\ |a[s]\rangle =$
for $i \in [0 \ldots s-1]$
$P\left(\frac{2\pi}{2^i}\right)|a_i\rangle$

The n-Qubit Transformation $Phase_f$

- The *Phase* program is used as a subroutine in a program that implements the n-qubit transformation

$$Phase_f : |x\rangle \rightarrow \exp\left(2\pi \boldsymbol{i} \frac{f(x)}{2^s}\right)|x\rangle.$$

- We implement this as follows.

define $Phase_f\ |x[k]\rangle =$

qubit $a[s]$	an s-bit temporary register	(1)
$U_f\|x\rangle\|a\rangle$	compute f in a	(2)
$Phase\ \|a\rangle$	perform phase shift by $\frac{2\pi a}{2^s}$	(3)
$U_f^{-1}\|x\rangle\|a\rangle$	uncompute f	(4)

Comments

- After Step (2), register a is entangled with x and contains the binary expansion of the angle $\phi(x)$ for the desired phase shift for the basis vector $|x\rangle$.
- Since registers a and x are entangled, changing the phase in register a during Step (3) is equivalent to changing the phase in register x.
- Step (4) uncomputes U_f to remove this entanglement so that the contents of register x end up in the desired state, no longer entangled with the temporary qubits.

State-Dependent Single-Qubit Amplitude Shifts

Aim: Efficiently approximate, to accuracy s, rotating each term in a superposition by a single-qubit rotation $R(\beta(x))$, where the angle $\beta(x)$ depends on the quantum state in another register.

- More specifically, we wish to implement a transformation that takes

$$|x\rangle \otimes |b\rangle \rightarrow |x\rangle \otimes (R(\beta(x))|b\rangle),$$

where:

  - $\beta(x) \approx f(x)\frac{2\pi}{2^s}$;
  - The approximating function $f : \mathbb{Z}_n \rightarrow \mathbb{Z}_s$ is efficiently computable.

- From an efficient implementation of U_f, we can implement the transformation Rot_f in $O(s)$ steps plus two calls to U_f.

Transformation *Rot*

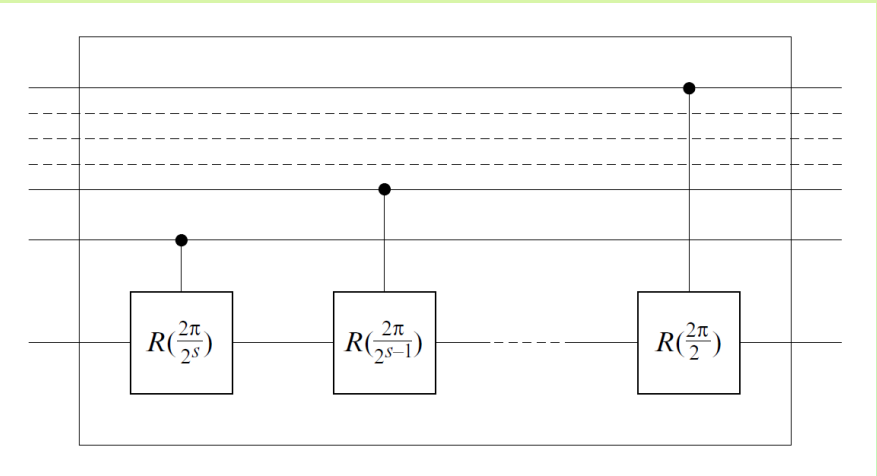
- The subroutine Rot_f uses an auxiliary transformation *Rot*.
- *Rot* shifts the amplitude in qubit b by the amount specified in register a,

$$|a\rangle \otimes |b\rangle \rightarrow |a\rangle \otimes \left(R\left(a\frac{2\pi}{2^s} \right) |b\rangle \right),$$

  where the contents of the s-qubit register a give the angle by which to rotate up to accuracy 2^{-s}.

- Using our program notation, this transformation can be described more concisely by

define *Rot* $|a[s]\rangle|b[1]\rangle =$
  for $i \in [0 \ldots s-1]$
    $|a_i\rangle$ **control** $R\left(\frac{2\pi}{2^i}\right)|b\rangle$.



The Rotation Rot_f

- The desired rotation specified by the function f can be achieved by the program:

define $Rot_f\ |x[k]\rangle|b[1]\rangle =$

qubit $a[s]$	an s-bit temporary register
$U_f\|x\rangle\|a\rangle$	compute f in a
$Rot\|a,b\rangle$	perform rotation by $\frac{2\pi a}{2^s}$
$U_f^{-1}\|x\rangle\|a\rangle$	uncompute f

Subsection 5

A Few Simple Quantum Algorithms

Deutsch-Jozsa Problem

- A function f is **balanced** if an equal number of input values to the function return 0 and 1.

  Deutsch-Jozsa Problem: Given a function $f : \mathbb{Z}_{2^n} \to \mathbb{Z}_2$ that is known to be either constant or balanced, and a quantum oracle

  $$U_f : |x\rangle|y\rangle \to |x\rangle|y \oplus f(x)\rangle$$

  for f, determine whether the function f is constant or balanced.

Deutsch-Jozsa Problem (First Step)

- The algorithm begins by using the phase change subroutine to negate terms of the superposition corresponding to basis vectors $|x\rangle$ with $f(x) = 1$.
- The subroutine returns the state

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} (-1)^{f(i)} |i\rangle.$$

- The subroutine uses a temporary qubit in state $|-\rangle$.
- It is vital that the subroutine end with that qubit unentangled with any other qubits.
- After disentanglement, the temporary qubit may be safely ignored.

Deutsch-Jozsa Problem (Second Step)

- Next, apply the Walsh transform W to the resulting state $|\psi\rangle$.
- We calculate

$$\begin{aligned}
|\phi\rangle &= W(|\psi\rangle) \\
&= W\left(\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{f(i)}|i\rangle\right) \\
&= \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{f(i)}W(|i\rangle) \\
&= \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}\left((-1)^{f(i)}\frac{1}{\sqrt{N}}\sum_{j=0}^{N-1}(-1)^{i\cdot j}|j\rangle\right) \\
&= \frac{1}{N}\sum_{i=0}^{n-1}\left((-1)^{f(i)}\sum_{j=0}^{N-1}(-1)^{i\cdot j}|j\rangle\right).
\end{aligned}$$

Deutsch-Jozsa Problem (Constant Case)

- Now suppose f is constant.
- Then $(-1)^{f(i)} = (-1)^{f(0)}$ is simply a global phase.
- So the state $|\phi\rangle$ is simply $|0\rangle$,

$$\begin{aligned} |\phi\rangle &= (-1)^{f(0)} \frac{1}{2^n} \sum_{j \in \mathbb{Z}_2^n} \left(\sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot j} \right) |j\rangle \\ &= (-1)^{f(0)} \frac{1}{2^n} \sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot 0} |0\rangle \\ &\quad \left(\sum_{i \in \mathbb{Z}_2^n} (-1)^{i \cdot j} = 0, \text{ for } j \neq 0 \right) \\ &= (-1)^{f(0)} |0\rangle. \end{aligned}$$

Deutsch-Jozsa Problem (Balanced Case)

- Next, suppose f balanced.
- Let

$$X_0 = \{x : f(x) = 0\}.$$

- Then we have

$$|\phi\rangle = \frac{1}{2^n} \sum_{j \in \mathbb{Z}_2^n} \left(\sum_{i \in X_0} (-1)^{i \cdot j} - \sum_{i \notin X_0} (-1)^{i \cdot j} \right) |j\rangle.$$

- In this case, for $j = 0$, the amplitude is zero,

$$\sum_{j \in X_0} (-1)^{i \cdot j} - \sum_{j \notin X_0} (-1)^{i \cdot j} = 0.$$

Deutsch-Jozsa Problem (Conclusion)

- Thus, measurement of state $|\phi\rangle$ in the standard basis will return:
  - $|0\rangle$, with probability 1, if f is constant;
  - A non-zero $|j\rangle$, with probability 1, if f is balanced.
- This quantum algorithm solves the Deutsch-Jozsa problem with a single evaluation of U_f.
- Any classical algorithm must evaluate f at least $2^{n-1}+1$ times to solve the problem with certainty.
- Thus, there is an exponential separation between the query complexity of this quantum algorithm and the query complexity for any possible classical algorithm that solves that problem with certainty.
- On the other hand, there are classical algorithms that solve this problem in fewer evaluations, but only with high probability of success.

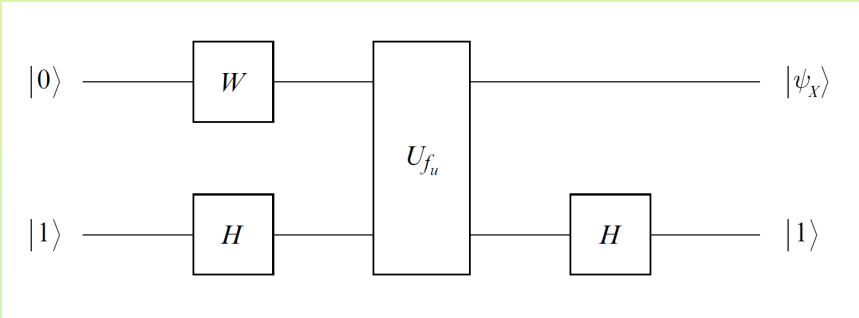
Bernstein-Vazirani Problem

Bernstein-Vazirani Problem: Determine the value of an unknown bit string u of length n, where only queries of the form $q \cdot u$, for some query string q, are allowed.

- The best classical algorithm uses $O(n)$ calls to $f_u(q) = q \cdot u \mod 2$.
- A quantum algorithm, closely related to the algorithm for the Deutsch-Jozsa problem, can find u in just a single call to U_{f_u}.
- On a quantum computer it is possible to determine u exactly with a single query (in superposition).

Bernstein-Vazirani Problem (Cont'd)

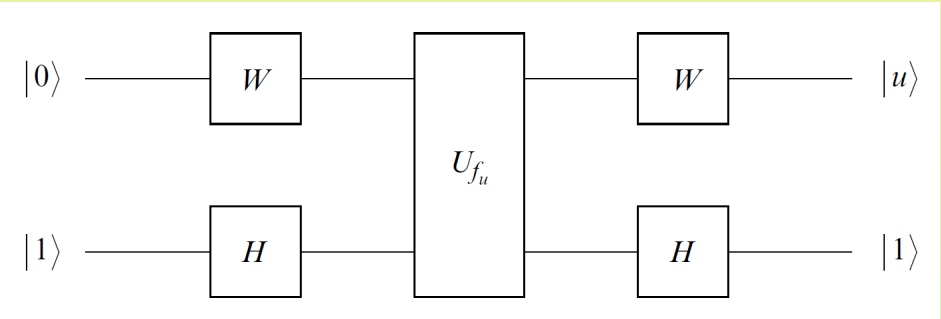
- Let
$$f_u(q) = q \cdot u \mod 2.$$
- Let, also,
$$U_{f_u} : |q\rangle|b\rangle \mapsto |q\rangle|b \oplus f_u(q)\rangle.$$
- Recall that in the special case $\phi = \pi$, the phase change subroutine can be accomplished by the circuit



- Applying this circuit results in the state $|\psi_X\rangle$ in the first register, with
$$|\psi_X\rangle = \frac{1}{\sqrt{2^n}} \sum_q (-1)^{f_u(q)} |q\rangle = \frac{1}{\sqrt{2^n}} \sum_q (-1)^{u \cdot q} |q\rangle.$$

Bernstein-Vazirani Problem (Cont'd)

- Now, applying the Walsh-Hadamard transformation W to this state, produces the state $|u\rangle$.



- Recall that $W|x\rangle = \frac{1}{\sqrt{2^n}} \sum_z (-1)^{x \cdot z} |z\rangle$.
- Thus

$$\begin{aligned} W|\psi_x\rangle &= W\left(\frac{1}{\sqrt{2^n}} \sum_q (-1)^{u \cdot q} |q\rangle\right) \\ &= \frac{1}{\sqrt{2^n}} \sum_q (-1)^{u \cdot q} W|q\rangle \\ &= \frac{1}{2^n} \sum_q (-1)^{u \cdot q} \left(\sum_z (-1)^{q \cdot z} |z\rangle\right). \end{aligned}$$

Bernstein-Vazirani Problem (Cont'd)

- Continuing, we have

$$\begin{aligned} W|\psi_X\rangle &= \frac{1}{2^n}\sum_q(-1)^{u\cdot q}\left(\sum_z(-1)^{q\cdot z}|z\rangle\right) \\ &= \frac{1}{2^n}\sum_q\sum_z(-1)^{(u\oplus z)\cdot q}|z\rangle) \\ &= \frac{1}{2^n}\sum_z\sum_q(-1)^{(u\oplus z)\cdot q}|z\rangle) \\ &= |u\rangle. \quad \left(\sum_q(-1)^{(u\oplus z)\cdot q}=0, \text{ if } u\oplus z\neq 0\right) \end{aligned}$$

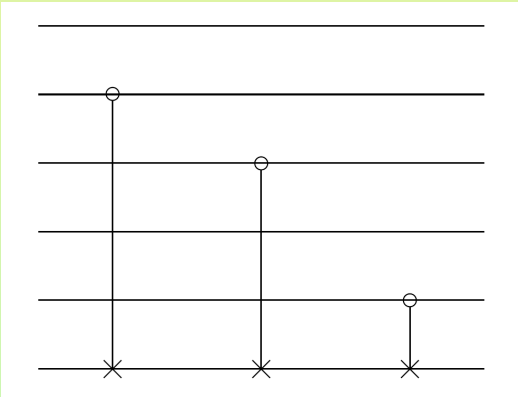
- Thus, measurement in the standard basis gives $|u\rangle$ with certainty.

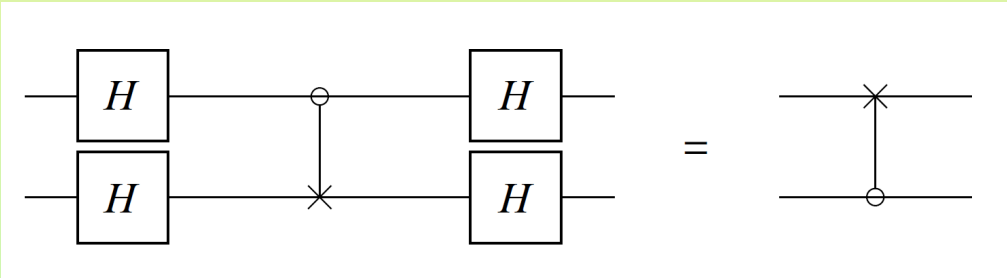
A Simpler Explanation

- A common explanation for how quantum algorithms works involves:
  - Using quantum parallelism, compute on all possible inputs at the same time;
  - Then cleverly manipulate the resulting superposition.
- The description we gave for the Bernstein-Vazirani algorithm fits this framework.
- There is a question, however, as to whether quantum parallelism is the right way of looking at algorithms.
- To illustrate this point, we give an alternative description, due to Mermin, of exactly this same algorithm.

A Simpler Explanation (Cont'd)

- The key to Mermin's explanation of the algorithm is to look at the circuit in the Hadamard basis.
- Consider the quantum black box for U_{f_u} in the Hadamard basis.
- It behaves as if consisting of a circuit containing a C_{not} from qubit i to the ancilla if and only if the ith bit of u is 1.

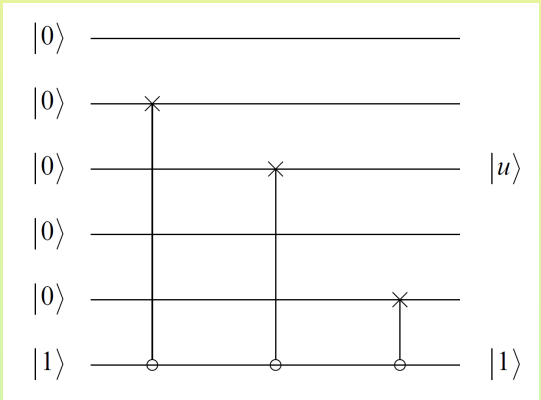




- Recall that Hadamard operations reverse the control and target roles of the qubits.

A Simpler Explanation (Cont'd)

- The Bernstein-Vazirani algorithms consists of starting with the state $|0\ldots0\rangle|1\rangle$ and applying Hadamard transformations to every qubit before and after the call to the black box for U_{f_u}.
- Thus, the Bernstein-Vazirani algorithm behaves as if it were a circuit consisting only of C_{not} operations from the ancilla qubit to the qubits corresponding to 1-bits of u.



- From this view of the circuit, it is immediate that the qubits end up in the state $|u\rangle$.
- This much simpler explanation, not speaking of quantum parallelism or of "computing on all possible inputs", is the right way to look at the algorithm.

Simon's Problem

Simon's Problem: Given a 2-to-1 function f such that, for some "hidden" $a \in \mathbb{Z}_2^n$,

$$f(x) = f(x \oplus a), \quad \text{for all } x \in \mathbb{Z}_2^n,$$

find the hidden string $a \in \mathbb{Z}_2^n$.

- Simon describes a quantum algorithm that can find a in only $O(n)$ calls to U_f, followed by $O(n^2)$ additional steps.
- The best a classical algorithm can do is $O(2^{n/2})$ calls to f.
- Simon's algorithm suggested to Shor an approach to the factoring problem, now known as Shor's algorithm.
- When we study Shor's algorithm, we will see that there are structural similarities between Shor's algorithm and Simon's algorithm.

Simon's Problem (Cont'd)

- To determine a, create the superposition $\sum_x |x\rangle|f(x)\rangle$.
- Suppose we measure the right part of the register.
- Let $f(x_0)$ be the measured value.
- This projects the state of the left register to

$$\frac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle).$$

- Applying the Walsh-Hadamard transformation W leads to

$$\begin{aligned}
&W\left(\tfrac{1}{\sqrt{2}}(|x_0\rangle + |x_0 \oplus a\rangle)\right) \\
&= \tfrac{1}{\sqrt{2}}\left(\tfrac{1}{\sqrt{2^n}}\textstyle\sum_y((-1)^{x_0\cdot y} + (-1)^{(x_0\oplus a)\cdot y})|y\rangle\right) \\
&= \tfrac{1}{\sqrt{2^{n+1}}}\textstyle\sum_y(-1)^{x_0\cdot y}(1 + (-1)^{a\cdot y})|y\rangle \\
&= \tfrac{2}{\sqrt{2^{n+1}}}\textstyle\sum_{y\,\cdot\, a \text{ even}}(-1)^{x_0\cdot y}|y\rangle.
\end{aligned}$$

Simon's Problem (Cont'd)

- Measurement of this state results in a random y, such that

$$y \cdot a = 0 \mod 2.$$

- So the unknown bits a_i of a must satisfy the equation

$$y_0 \cdot a_0 \oplus \cdots \oplus y_{n-1} \cdot a_{n-1} = 0.$$

- This computation is repeated until n linearly independent equations have been found.
- Each time the computation is repeated, the resulting equation has at least a 50 percentage chance of being linearly independent of the previous equations obtained.

Simon's Problem (Cont'd)

- After repeating the computation 2^n times, there is a 50 percentage chance that n linearly independent equations have been found.
- These equations can be solved to find a in $O(n^2)$ steps.
- Thus, with high likelihood, the hidden string a will be found using:
  - $O(n)$ calls to U_f;
  - $O(n^2)$ steps to solve the resulting set of equations.

Distributed Computation

The Problem: Let $N = 2^n$.

Alice and Bob are each given an N-bit number, u and v, respectively.

The objective is for Alice to compute an n-bit number a and Bob to compute an n-bit number b such that

$$\begin{aligned} d_H(u,v) = 0 &\rightarrow a = b \\ d_H(u,v) = \frac{N}{2} &\rightarrow a \neq b \\ \text{else} &\rightarrow \text{no condition on } a \text{ and } b \end{aligned}$$

where $d_H(u,v)$ is the Hamming distance between u and v.

- In other words, Alice and Bob need an algorithm that produces a and b from any u and v such that:
  - If $u = v$, then $a = b$;
  - If u and v differ in half of their bits, then $a \neq b$;
  - If the Hamming distance of u and v is anything other than 0 or $\frac{N}{2}$, a and b can be anything.

Distributed Computation (Non-Triviality)

- This problem is nontrivial because u and v are exponentially larger than a and b.
- Given a sufficient supply of entangled pairs, this problem can be solved without additional communication between Alice and Bob.
- On the other hand, a classical solution requires communication of at least $\frac{N}{2}$ bits between the two parties.

Distributed Computation (Set-Up)

- Suppose Alice and Bob share n entangled pairs of particles (a_i, b_i), each in state

$$\frac{1}{\sqrt{2}}(|00\rangle + |11\rangle),$$

where:

  - Alice can access particles a_i;
  - Bob can access particles b_i.

- We write the state of the $2n$ particles making up these n entangled pairs in order $a_0, a_1, \ldots, a_{n-1}, b_0, b_1, \ldots, b_{n-1}$.
- So the entire $2n$-qubit state is written

$$\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i, i\rangle,$$

where:

  - Alice can manipulate the first n qubits;
  - Bob can manipulate the last n qubits.

Distributed Computation (Solution)

- The problem can be solved without additional communication.
- Alice applies the following:
  - Using the phase change subroutine, with $f(i) = u_i$, she performs
$$\sum |i\rangle \to \sum (-1)^{u_i} |i\rangle;$$
  - She then applies the Walsh transform W on her n qubits.
- Bob applies the following:
  - Using the phase change subroutine, with $f(i) = v_i$, he performs
$$\sum |i\rangle \to \sum (-1)^{v_i} |i\rangle;$$
  - He then applies the Walsh transform W on his n qubits.
- Together their particles are now in the common global state
$$|\psi\rangle = W\left(\frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} (-1)^{u_i \oplus v_i} |i\rangle |i\rangle\right).$$

Distributed Computation (Solution Cont'd)

- Alice and Bob now measure their respective part of the state to obtain results a and b.
- We need to show that a and b have the desired properties.
- The probability that measurement results in $a = x = b$ is

$$|\langle x, x|\psi\rangle|^2.$$

- We must show that this probability is

$$\begin{cases} 1, & \text{if } u = v, \\ 0, & \text{if } d_H(u, v) = \frac{N}{2}. \end{cases}$$

Distributed Computation (Solution Cont'd)

- Use $W^{(\ell)}$ indicates that W is acting on an ℓ qubit state.
- Simplify the state

$$\begin{aligned}
|\psi\rangle &= W^{(2n)}\left(\frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{u_i\oplus v_i}|i\rangle|i\rangle\right)\\
&= \frac{1}{\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{u_i\oplus v_i}(W^{(n)}|i\rangle\otimes W^{(n)}|i\rangle)\\
&= \frac{1}{N\sqrt{N}}\sum_{i=0}^{N-1}\sum_{j=0}^{N-1}\sum_{k=0}^{N-1}(-1)^{u_i\oplus v_i}(-1)^{i\cdot j}(-1)^{i\cdot k}|jk\rangle.
\end{aligned}$$

- Now we obtain

$$\begin{aligned}
\langle x,x|\psi\rangle &= \frac{1}{N\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{u_i\oplus v_i}(-1)^{i\cdot x}(-1)^{i\cdot x}\\
&= \frac{1}{N\sqrt{N}}\sum_{i=0}^{N-1}(-1)^{u_i\oplus v_i}.
\end{aligned}$$

Distributed Computation (Solution Cont'd)

- We found

$$\langle x, x|\psi\rangle = \frac{1}{N\sqrt{N}} \sum_{i=0}^{N-1} (-1)^{u_i \oplus v_i}.$$

  - Suppose $u = v$.
    Then $(-1)^{u_i \oplus v_i} = 1$. Consequently, $\langle x, x|\psi\rangle = \frac{1}{\sqrt{N}}$.
    So the probability $|\langle x, x|\psi\rangle|^2 = \frac{1}{N}$.
    The probability, summed over the N possible values of x, is 1.
    So when Alice and Bob measure they obtain, with probability 1, states a and b with $a = b = x$ for some bit string x.
  - Suppose $d_H(u, v) = \frac{N}{2}$
    The sum

$$\langle x, x|\psi\rangle = \frac{1}{N\sqrt{N}} \sum_{i=0}^{N-1} (-1)^{u_i \oplus v_i}$$

    has an equal number of $+1$ and -1 terms. So $\langle x, x|\psi\rangle = 0$.
    Thus, Alice and Bob measure the same value with probability 0.

Subsection 6

Comments on Quantum Parallelism

Two Important Misconceptions

- Quantum parallelism's role in quantum computation has often been misunderstood.
- So it is worth addressing some common misconceptions.
- Consider the notation $\frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x, f(x)\rangle$.
- It suggests that exponentially more computation is being done by the quantum operation U_f acting on the superposition $\sum_x |x, 0\rangle$ than by a classical computer computing $f(x)$ from x.
  - The next paragraph explains how this view is misleading and how it does not explain the power of quantum computation.
- The exponential size of the n-qubit quantum state space may seem to suggest that an exponential speedup over the classical case can always be obtained using quantum parallelism.
  - This statement is generally incorrect, although in certain special cases quantum computation does provide such speedups.

On the Exponential Speedup

- Recall that only one input/output pair can be extracted by measurement in the standard basis from the superposition generated quantum parallelism.
- It is not possible to extract more input/output pairs in any other way, since only m bits of information can be extracted from an m-qubit state.
- Thus, while the 2^n values of $f(x)$ appear in the single superposition state, it still takes 2^n computations of U_f to obtain them all, no better than the classical case.

On the Exponential Speedup (Cont'd)

- This limitation leaves open the possibility that any classical algorithm that takes 2^n steps to obtain n bits of output could be done in a single step on a quantum computer.
- Some algorithms do give speedups of this magnitude over classical algorithms.
- However, the optimality of Grover's algorithm (to be shown) shows that there are problems of this form for which it is known that no quantum algorithm can provide an exponential speedup.
- Furthermore, lower bound results exist that show that, for many problems, quantum computation cannot provide any speedup at all.
- Thus, quantum parallelism and quantum computation do not, in general, provide the exponential speedup suggested by the notation.

The State Space Approximation Limitations

- A superposition like $\frac{1}{\sqrt{N}} \sum |x, f(x)\rangle$ is still only a single state of the quantum state space.
- The n-qubit quantum state space is extremely large, so large that the vast majority of states cannot even be approximated by an efficient quantum algorithm.
- Thus, an efficient quantum algorithm cannot even come close to most states in the state space.
- For this reason, quantum parallelism does not, and efficient quantum algorithms cannot, make use of the full state space.
- As Mermin's explanation of the Bernstein-Vazirani algorithm illustrates, even when quantum parallelism can be used to describe an algorithm, it is not necessarily correct to view it as key to the algorithm.

Techniques for Manipulating the State

- Many algorithms are described in terms of quantum parallelism.
- The heart of the algorithm is the way in which the algorithm manipulates the state generated by quantum parallelism.
- This sort of manipulation has no classical analog.
- It usually requires nontraditional programming techniques.
- We list a couple of general techniques.

Techniques for Manipulating the State

- **Amplify output values of interest**:

  The general idea is to transform the state in such a way that values of interest have:

  - A larger amplitude;
  - Therefore, a higher probability of being measured.

  Grover's algorithm (to be seen) exploits this approach.

- **Find properties of the set of all the values of** $f(x)$:

  This idea is exploited in Shor's algorithm (to be seen), which uses a quantum Fourier transformation to obtain the period of f.

  Other algorithms that take this approach are the ones given for:

  - The Deutsch-Jozsa Problem;
  - The Bernstein-Vazirani Problem;
  - Simon's Problem.

Subsection 7

Machine Models and Complexity Classes

Languages and Machines

- Computational complexity classes are defined in terms of a **language** and **machines** that recognize that language.
- In this section, the term **machine** refers to any quantum or classical computing device that runs a single algorithm on which we can count the number of computation steps and storage cells used.
- A **language** L over an alphabet Σ is a subset of the finite strings Σ^* of elements from Σ.
- A language L is **recognized** by a machine M if, for each string $x \in \Sigma^*$, the machine M can *determine* if $x \in L$.
- Exactly what **determine** means depends on the kind of machine we are considering.

  Example: Given input x, a classical deterministic machine may answer **Yes**, $x \in L$, or **No**, $x \notin L$, or it may never halt.

  Probabilistic and quantum machines might answer **Yes** or **No** correctly with certain probabilities.

Classical Machines and Quantum Analogs

- We consider five kinds of classical machines:
  - Deterministic (**D**);
  - Nondeterministic (**N**);
  - Randomized (**R**);
  - Probabilistic (**Pr**);
  - Bounded probability of error (**BP**).
- Each of these types of classical machine has a quantum analog (**EQ**, **NQ**,**RQ**, **PrQ**,**BQ**).
- Of particular interest will be:
  - Quantum deterministic (exact) machines (**EQ**);
  - Quantum bounded probability of error machines (**BQ**).
- We use these types of machine to define numerous complexity classes of varying resource constraints.

Deterministic Machines

- For each kind of machine M, there is a single language L_M that M recognizes.
- A machine is **deterministic** if, whenever it sometimes answers **Yes** on a given input x, it always answers **Yes** on that input.
- A deterministic machine D **recognizes** the language

$$L_D = \{x \in \Sigma^* : D(x) = \textbf{Yes}\} = \{x : P(D(x) = \textbf{Yes}) = 1\}.$$

- By definition of deterministic, the probability

$$P(D(x) = \textbf{Yes}) = 0, \quad \text{for all } x \notin L.$$

Bounded Probability of Error Machines

- A **bounded probability of error** machine, acting on a given input x, either answers **Yes** with probability at least $\frac{1}{2} + \epsilon$ or with probability no more than $\frac{1}{2} - \epsilon$.
- Given a bounded probability of error machine BP,

$$L_{BP} = \left\{ x : P(BP(x) = \textbf{Yes}) \geq \frac{1}{2} + \epsilon \right\}.$$

- For $x \notin L_{BP}$,

$$P(BP(x) = \textbf{Yes}) \leq \frac{1}{2} - \epsilon.$$

Conditions on Various Types of Machines

- A machine may not give an answer at all for some inputs.
- The table summarizes the conditions for the various types of machines we consider.

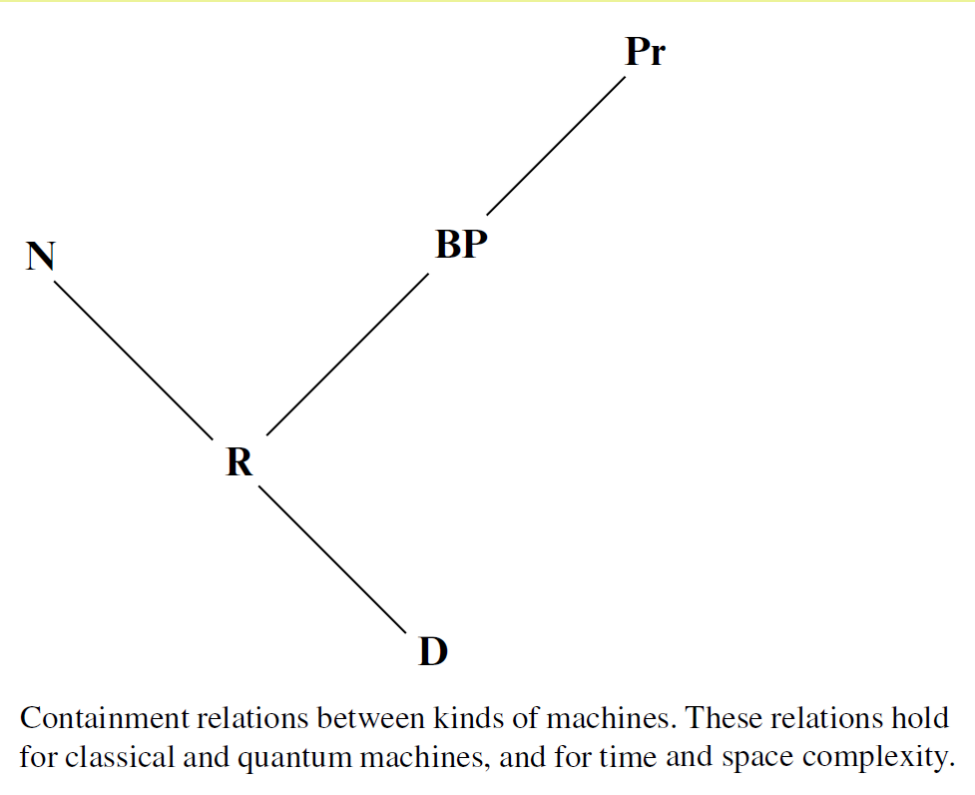
Probability for a particular kind of machine to answer *Yes* when given an input x that is or is not and element of language L

Prefix	Kind of machine	$P(x \in L)$	$P(x \notin L)$
Classical			
D	Deterministic	$= 1$	$= 0$
N	Nondeterministic	> 0	$= 0$
R	Randomized (Monte Carlo)	$> \frac{1}{2} + \epsilon$	$= 0$
Pr	Probabilistic	$> \frac{1}{2}$	$\leq \frac{1}{2}$
BP	Bounded probability of error	$> \frac{1}{2} + \epsilon$	$\leq \frac{1}{2} - \epsilon$
Quantum			
EQ	Quantum deterministic (exact)	$= 1$	$= 0$
BQ	Quantum bounded probability of error	$> \frac{1}{2} + \epsilon$	$\leq \frac{1}{2} - \epsilon$

- The quantum machine types recognize a language with the same probability as their classical counterparts.

Relations Between the Machines

- The next figure illustrates containment relations between the kinds of machines.



Containment relations between kinds of machines. These relations hold for classical and quantum machines, and for time and space complexity.

- Containment means that, e.g., by definition, each **D** machine is also an **R** machine.

The Role of ϵ

- A language is recognized by a kind of machine if there exists a machine of that kind that recognizes it.
- The set of languages recognized by the types of machines we have defined does not depend on the particular value of ϵ.

  Example: Suppose we are given a **Pr** machine M that answers **Yes**, for $x \in L$, with probability $P(x \in L) > \frac{1}{2} + \epsilon$.

  We can construct a new **Pr** machine M' that:
  - Runs M three times;
  - Answers **Yes** if M answers **Yes** at least two times.

  Then M' will accept $x \in L$ with probability $> \frac{1}{2} + \frac{3}{2}\epsilon - \epsilon^3$.

The Role of ϵ (Cont'd)

- Since the exact value of ϵ is not important, some authors use a fixed value, such as $\epsilon = \frac{1}{4}$.
- However, the case $P(x \in L) > \frac{1}{2}$ is different from $P(x \in L) > \frac{1}{2} + \epsilon$.
- In the former case no polynomial number of repetitions can guarantee an increase in the success probability above a given threshold $\frac{1}{2} + \epsilon$.

Time and Space Complexity

- One is concerned about the probability that a machine answer correctly.
- But complexity theory is also concerned about quantifying the amount of resources, particularly time and space, that a machine uses to obtain its answers.
- A machine recognizes a language L **in time** $O(f)$ if, for any string $x \in \Sigma^*$ of length n, it answers **Yes** or **No** within $t(n)$ steps and $t \in O(f)$.
- A machine recognizes a language L **in space** $O(f)$ if, for any string $x \in \Sigma^*$ of length n, it answers **Yes** or **No** using at most $s(n)$ storage units, measured in bits or qubits, where $s \in O(f)$.

Complexity Classes

- A **complexity class** is the set of languages recognized by a particular kind of machine within given resource bounds.
- Specifically, for $\mathbf{m} \in \{\mathbf{D}, \mathbf{EQ}, \mathbf{N}, \mathbf{R}, \mathbf{Pr}, \mathbf{BP}\}$, we consider the classes $\mathbf{mTime}(f)$ and $\mathbf{mSpace}(f)$.
- Language L is in complexity class $\mathbf{mTime}(f)$ if there exists a machine M of kind $\mathbf{m}$ that recognizes L in time $O(f)$.
- Language L is in complexity class $\mathbf{mSpace}(f)$ if there exists a machine M of kind $\mathbf{m}$ that recognizes L in space $O(f)$.

List of Complexity Classes

- We are particularly interested in machines that use only a polynomial amount of resources, and to a lesser extent in those that use only an exponential amount.
- For example, we are interested in the class $\mathbf{P} = \mathbf{DTime}(n^k)$ of machines that respond to an input of length n using only $O(n^k)$ time, for some k.
- The following shorthand notations are common:

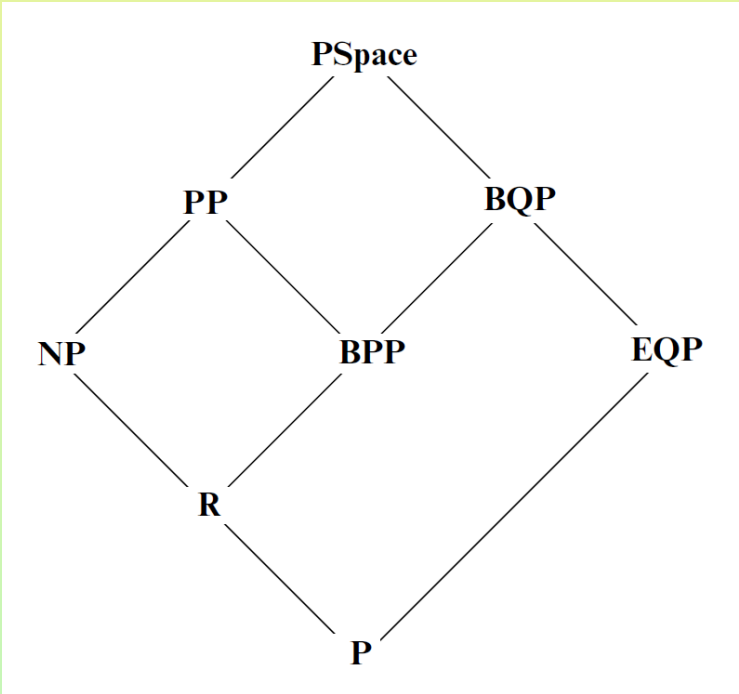
P	$\mathbf{DTime}(n^k)$	**EQP**	$\mathbf{EQTime}(n^k)$
NP	$\mathbf{NTime}(n^k)$	**R**	$\mathbf{RTime}(n^k)$
PP	$\mathbf{PrTime}(n^k)$	**BPP**	$\mathbf{BPTime}(n^k)$
BQP	$\mathbf{BQTime}(n^k)$	**PSpace**	$\mathbf{DSpace}(n^k)$
NPSpace	$\mathbf{NSpace}(n^k)$	**EXP**	$\mathbf{DTime}(k^n)$

Time Complexity and Halting

- For time classes, we can assume that machines always halt, since the function f provides an upper bound on the possible runtimes.
- However, machines in the space complexity classes may never halt on some inputs.
- Therefore, we define $\mathbf{m}_H\mathbf{Space}(f)$ to be the class of languages that are recognized by a halting machine of type $\mathbf{m}$ in space $O(f)$.
- Obviously, $\mathbf{m}_H\mathbf{Space}(f) \subseteq \mathbf{mSpace}(f)$.
- Note that in the circuit model all computations will terminate.
- Analysis of the complexity of nonhalting space classes requires a different model of computation, such as quantum Turing machines.

Relations Between Complexity Classes

- We give informal arguments for some of the containment relations involving quantum complexity classes.
- The figure depicts the known containment relations involving classical and quantum time complexity classes.



- Nothing is known about the relation between **BQP** and **NP** or **PP**.

$\mathbf{P} \subseteq \mathbf{EQP}$ and $\mathbf{EQP} \subseteq \mathbf{BQP}$

$\mathbf{P} \subseteq \mathbf{EQP}$: Consider a classical polynomial time computation.

It can be performed by a polynomial size circuit family.

The circuit family can be made to operate reversibly, with only a slight increase in time and space.

Any reversible polynomial time algorithm can be turned into a polynomial time exact quantum algorithm.

$\mathbf{EQP} \subseteq \mathbf{BQP}$: This containment is trivial.

Every exact quantum algorithm has bounded probability of error.

$\mathbf{BPP} \subseteq \mathbf{BQP}$

BPP $\subseteq$ **BQP**: Consider a computation performed by a machine M in **BPP**.

It can be approximated arbitrarily closely by a machine $\widetilde{M}$ that makes a single equiprobable binary decision at each step.

The decision tree is of polynomial depth.

So a sequence of choices can be encoded by a polynomial size bit string c.

Construct a deterministic machine $\widetilde{M}_d$ that, when applied to c and x, will perform the same computation as $\widetilde{M}$ applied to x making the random choices c.

$\mathbf{BPP} \subseteq \mathbf{BQP}$ (Cont'd)

- Construct a polynomial time quantum machine $\widetilde{M}_q$ that can be applied to the superposition of all possible random choices c applied to x, $\sum_c |c, x, 0\rangle$, producing $\sum_c |c, x, \widetilde{M}_d(c, x)\rangle$.

  In effect, $\widetilde{M}_q$ performs all possible computations of $\widetilde{M}$ on x in parallel.

  The probability of reading an accepting answer from $\widetilde{M}_q$ is the same as the probability that $\widetilde{M}$ would accept x.
- It is not known whether $\mathbf{BPP} \subseteq \mathbf{BQP}$ is a proper inclusion.
- In fact, showing $\mathbf{BPP} \neq \mathbf{BQP}$ would answer the open question as to whether $\mathbf{BPP} = \mathbf{PSpace}$.

BQP ⊆ PSpace

BQP ⊆ PSpace: Consider a machine in **BQP**, acting on an input of size n, that:

- Starts from a known state $|\psi_0\rangle = |0\rangle$;
- Proceeds for k steps;
- Ends with a measurement.

We show that such a machine can be approximated arbitrarily closely, in the sense of computing any amplitude of the final state to a specified precision, in polynomial space.

Suppose the state after step i is

$$|\psi_i\rangle = \sum_j a_{ij}|j\rangle.$$

Each state $|\psi_i\rangle$, $i \neq 0$, may be a superposition of an exponential (in n) number of basis vectors.

It is possible, using space polynomial in n, to compute the amplitude a_{kj} of an arbitrary basis vector in the final superposition $|\psi_k\rangle$.

BQP ⊆ PSpace (Cont'd)

- We may assume each step corresponds to a primitive quantum gate U_i that operates on at most $d \leq 3$ quantum bits.

  For these transformations, we show that the amplitude $a_{i+1,j}$ of basis vector $|j\rangle$ in state $|\psi_{i+1}\rangle$ depends only on the amplitudes $a_{i,j}$ of the small number ($2^d \leq 8$) of basis vectors of the preceding state $|\psi_i\rangle$ that differ from $|j\rangle$ only in the bits that are being operated on by the gate.

  Without loss of generality, assume that $U = U_{i+1}$ operates on the last d quantum bits.

  We will use the the shorthand $x \circ y$ to stand for $2^d x + y$.

  Let

  $$u_{qr} = \langle r|U|q\rangle,$$

  for basis elements $|r\rangle$ and $|q\rangle$ in the standard basis for a 2^d-dimensional space.

BQP ⊆ PSpace (Cont'd)

- We have

$$\begin{aligned}
|\psi_{i+1}\rangle &= (I^{n-d} \otimes U)|\psi_i\rangle \\
&= \textstyle\sum_j a_{ij}(I^{n-d} \otimes U)|j\rangle \\
&= \textstyle\sum_{p=0}^{2^{n-d}-1} \sum_{q=0}^{2^d-1} a_{i,p\circ q}|p\rangle \otimes U|q\rangle \\
&= \textstyle\sum_p \sum_q a_{i,p\circ q}|p\rangle \otimes \sum_{r=0}^{2^d-1} u_{qr}|r\rangle \\
&= \textstyle\sum_p \sum_r (\sum_{q=0}^{2^d-1} u_{qr} a_{i,p\circ q})|p\rangle|r\rangle.
\end{aligned}$$

It follows that each amplitude

$$a_{i+1,p\circ r} = \sum_{q=0}^{2^d-1} u_{qr} a_{i,p\circ q}$$

depends only on 2^d amplitudes $a_{i,p\circ q}$ of the preceding state.

BQP ⊆ PSpace (Cont'd)

- By induction, we argue that it requires storage of $i2^d$ amplitudes to compute a single amplitude of state $|\psi_i\rangle$.

  Since we know $|\psi_0\rangle$, it takes no space to compute the amplitude $\langle j|\psi_0\rangle$ for any j.

  As we have just seen, the amplitude $a_{i+1,j}$ can be computed from 2^d amplitudes of $|\psi_i\rangle$.

  We can do this by computing each of these amplitudes in turn.

  This requires:

  - Storing at most $i2^d$ amplitude values;
  - Storing the resulting 2^d amplitudes;
  - Computing $a_{i+1,j}$.

  Overall, this process requires storage of $(i+1)2^d$ amplitude values.

BQP $\subseteq$ PSpace (Cont'd)

- Let M be the maximum precision required at any point in the computation to obtain the desired precision at the end.

  The total accumulated error is no larger than the sum of the errors of individual steps.

  Thus, the number M grows only linearly in the number of steps.

  Any one amplitude value can be stored in space M.

  The amplitude of any basis vector of the final superposition, after k steps, can be computed in $k2^d M$ space.
  - By assumption, k is polynomial in n.
  - d is a constant no more than 3.
  - M only grows linearly with k.

  So it takes only polynomial space to compute a single amplitude of the final state $|\psi_k\rangle$.

BQP $\subseteq$ **PSpace** (Cont'd)

- To simulate the algorithm, choose a basis vector $|j\rangle$ randomly.

  Alternatively, they could be taken in a specified order.

  Calculate the amplitude a_{kj}.

  Generate a random number between 0 and 1.

  Check whether it is less than $|a_{kj}|$.

  If it is, return $|j\rangle$.

  Otherwise:

  - Free all the space;
  - Choose another basis vector;
  - Repeat the process.

  Repeat as often as necessary until a basis vector is returned (time is not an issue!).

  In his way, any computation in **BQP** can be simulated classically in polynomial space.

Subsection 8

Quantum Fourier Transformations

Discrete Fourier Transform

- The **discrete Fourier transform** (**DFT**) operates on a discrete complex-valued function to produce another discrete complex-valued function.
- Consider a function

$$a : [0, \ldots, N-1] \to \mathbb{C}.$$

- The discrete Fourier transform produces a function

$$A : [0, \ldots, N-1] \to \mathbb{C},$$

defined by

$$A(x) = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a(k) \exp\left(2\pi \boldsymbol{i} \frac{kx}{N}\right).$$

Fourier Coefficients

- The discrete Fourier transform can be viewed as a linear transformation, taking column vector $(a(0), \ldots, a(N-1))^T$ to $(A(0), \ldots, A(N-1))^T$, with matrix representation F having entries

$$F_{xk} = \frac{1}{\sqrt{N}} \exp\left(2\pi \boldsymbol{i} \frac{kx}{N}\right), \quad x, k = 0, \ldots, N-1.$$

- The values $A(0), \ldots, A(N-1)$ are called the **Fourier coefficients** of the function a.

Example

- Let $a : [0, \ldots, N-1] \to \mathbb{C}$ be the periodic function

$$a(x) = \exp\left(-2\pi \boldsymbol{i} \frac{ux}{N}\right),$$

  for some frequency u evenly dividing N.
- We assume that the function is not constant, that is $0 < u < N$.
- The Fourier coefficients for this function are

$$\begin{aligned} A(x) &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} a(k) \exp\left(2\pi \boldsymbol{i} \frac{kx}{N}\right) \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \exp\left(-2\pi \boldsymbol{i} \frac{uk}{N}\right) \exp\left(2\pi \boldsymbol{i} \frac{kx}{N}\right) \\ &= \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} \exp\left(2\pi \boldsymbol{i} \frac{k(x-u)}{N}\right). \end{aligned}$$

- Sums of the form $\sum_{k=0}^{N-1} \exp\left(2\pi \boldsymbol{i} k \frac{r}{N}\right)$ vanish unless $r = 0 \mod N$.
- Since $u < N$, $A(x) = 0$ unless $x - u = 0$.
- Thus, only $A(u)$ will be non-zero.

Behavior of the Discrete Fourier Transform

- Any periodic complex-valued function a with period r and frequency $u = \frac{N}{r}$ can be approximated, using its Fourier series, as the sum of exponential functions whose frequencies are multiples of u.
- Since the Fourier transform is linear, the Fourier coefficients $A(x)$ of any periodic function will be the sum of the Fourier coefficients of the component functions.
- If r divides N evenly, the Fourier coefficients $A(x)$ will be non-zero only for those x that are multiples of $u = \frac{N}{r}$.
- If r does not divide N evenly, the result only approximates this behavior, with the highest values at the integers closest to multiples of $u = \frac{N}{r}$ and low values at integers far from these multiples.

Fast Fourier Transform

- The **fast Fourier transform** (**FFT**) is an efficient implementation of the discrete Fourier transform (DFT) when N is a power of two: $N = 2^n$.
- The key to the implementation is that $F^{(n)}$ can be recursively decomposed in terms of Fourier transforms for lower powers of 2.
- Let $\omega_{(n)}$ be the N-th root of unity, $\omega_{(n)} = \exp\left(\frac{2\pi \boldsymbol{i}}{N}\right)$.
- The entries of the $N \times N$ matrix $F^{(n)}$ for the $N = 2^n$ dimensional Fourier transform are simply

$$F_{ij}^{(n)} = \omega_{(n)}^{ij},$$

where we index the entries of all $N \times N$ matrices by $i \in \{0, \ldots, N-1\}$ and $j \in \{0, \ldots, N-1\}$.

Fast Fourier Transform (Cont'd)

- Let $F^{(k)}$ be the $2^k \times 2^k$ matrix for the 2^k-dimensional Fourier transform.
- Let $I(k)$ be the $2^k \times 2^k$ identity matrix.
- Let $D(k)$ be the $2^k \times 2^k$ diagonal matrix with elements

$$\omega_{(k+1)}^0, \ldots, \omega_{(k+1)}^{2^k-1}.$$

- Let $R^{(k)}$ be the permutation that maps the vector entries at index $2i$ to position i and at index $2i+1$ to position $i+2^{k-1}$.
- The entries of the $2^k \times 2^k$ matrix for $R^{(k)}$ are given by

$$R_{ij}^{(k)} = \begin{cases} 1, & \text{if } 2i = j, \\ 1, & \text{if } 2(i-2^{k-1})+1 = j, \\ 0, & \text{otherwise.} \end{cases}$$

Fast Fourier Transform (Cont'd)

- One may verify that

$$F^{(k)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k-1)} & D^{(k-1)} \\ I^{(k-1)} & -D^{(k-1)} \end{pmatrix} \begin{pmatrix} F^{(k-1)} & 0 \\ 0 & F^{(k-1)} \end{pmatrix} R^{(k)}.$$

- There exists an implementation of the fast Fourier transform, based on this recursive decomposition, that uses only $O(nN)$ steps.

The Quantum Fourier Transform

- The **quantum Fourier transform** (**QFT**) is a variant of the discrete Fourier transform, which, like the fast Fourier transform (FFT), assumes that $N = 2^n$.
- The amplitudes a_x of any quantum state $\sum_x a_x|x\rangle$ can be viewed as a function of x, which we will denote by $a(x)$.
- The quantum Fourier transform operates on a quantum state by sending

$$\sum_x a(x)|x\rangle \to \sum_x A(x)|x\rangle,$$

where:

  - $A(x)$ are the Fourier coefficients of the discrete Fourier transform of $a(x)$;
  - x ranges over the integers between 0 and $N-1$.

- If the state were measured in the standard basis right after the Fourier transform was performed, the probability that the resulting state would be $|x\rangle$ would be $|A(x)|^2$.

The Quantum Fourier Transform versus U_f

- The quantum Fourier transform generalizes from a classical complex-valued function in quite a different way from how U_f generalizes a binary classical function f.
- Here, there is no need for an additional output register.
- Instead, the output of the classical function is placed in the complex amplitudes of the final superposition state.

Properties of the QFT

- Suppose a state has amplitudes given by a periodic function $a(x) = a_x$ with period r, where r is a power of 2.
- Then the quantum Fourier transform $\sum_x A(x)|x\rangle$ of a would have $A(x)$ zero, except when x is a multiple of $\frac{N}{r}$.
- Suppose the state was measured in the standard basis at this point.
- The result would be one of the basis vectors $|x\rangle$ with label a multiple of $\frac{N}{r}$, say $|j\frac{N}{r}\rangle$.
- The quantum Fourier transform behaves in only approximately this way when the period is not a power of 2 (does not divide $N = 2^n$).
  - States labeled with integers near multiples of $\frac{N}{r}$ would be measured with high probability.
  - The larger the power of 2 used as a base for the transform, the closer the approximation.

Time Complexity of QFT

- The implementation of the quantum Fourier transform is based on that of the fast Fourier transform.
- However, the quantum Fourier transform can be implemented exponentially faster.
- It needs only $O(n^2)$ operations.
- In contrast, as was mentioned, the fast Fourier transform needs $O(nN)$ operations.

A Quantum Circuit for Fast Fourier Transform

- We show how to implement efficiently the quantum Fourier transform $U_F^{(n)}$ for $N = 2^n$, defined by

$$U_F^{(n)} : |k\rangle \to \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} \exp\left(\frac{2\pi \boldsymbol{i} kx}{N}\right)|x\rangle.$$

- The quantum Fourier transform for $N = 2$ is the familiar Hadamard transformation:

$$\begin{aligned} U_F^{(1)} : \quad |0\rangle &\to \tfrac{1}{\sqrt{2}} \textstyle\sum_{x=0}^{1} e^{0}|x\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle + |1\rangle), \\ |1\rangle &\to \tfrac{1}{\sqrt{2}} \textstyle\sum_{x=0}^{1} e^{\pi \boldsymbol{i} x}|x\rangle = \tfrac{1}{\sqrt{2}}(|0\rangle - |1\rangle). \end{aligned}$$

A Quantum Circuit for Fast Fourier Transform (Cont'd)

- We can compute $U_F^{(n)}$, using the recursive decomposition,

$$U_F^{(k+1)} = \frac{1}{\sqrt{2}} \begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix} \begin{pmatrix} U_F^{(k)} & 0 \\ 0 & U_F^{(k)} \end{pmatrix} R^{(k+1)}.$$

- All of the component matrices are unitary (the multiplicative factor in front goes with the first matrix).
- It remains to be shown how these components can be efficiently realized on a quantum computer.

Quantum Implementation ($R^{(k+1)}$)

1. We can write the rotation $R^{(k+1)}$ as

$$R^{(k+1)} = \sum_{i=0}^{2^k-1} |i\rangle\langle 2i| + |i+2^k\rangle\langle 2i+1|.$$

- It can be accomplished by a simple permutation of the $k+1$ qubits.
  - Qubit 0 becomes qubit k;
  - Qubits 1 through k become qubits 0 through $k-1$.
- This permutation can be implemented using $k-1$ swap operations.

Quantum Implementation $(I \otimes U_F^{(k)})$

2. Consider, next, the transformation

$$\begin{pmatrix} U_F^{(k)} & 0 \\ 0 & U_F^{(k)} \end{pmatrix} = I \otimes U_F^{(k)}.$$

- It can be implemented by recursively applying the quantum Fourier transform to qubits 0 through k.

Quantum Implementation ($D^{(k)}$)

3. For $k \geq 1$, the $2^k \times 2^k$-diagonal matrix of phase shifts $D^{(k)}$ can be recursively decomposed as

$$D^{(k)} = D^{(k-1)} \otimes \begin{pmatrix} 1 & 0 \\ 0 & \omega_{(k+1)} \end{pmatrix}.$$

- We recursively decomposing $D^{(k)}$ in this way.
- Then $D^{(k)}$ can be implemented by applying

$$\begin{pmatrix} 1 & 0 \\ 0 & \omega_{(i+1)} \end{pmatrix}$$

to qubit i, for $1 \leq i \leq k$.

- Thus altogether $D^{(k-1)}$ can be implemented using k single-qubit gates.

Quantum Implementation (Final Transformation)

4. Given the implementation of $D^{(k)}$, consider, finally,

$$\frac{1}{\sqrt{2}}\begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix}.$$

- This can also be implemented with only k gates.
- We have

$$\begin{aligned} \frac{1}{\sqrt{2}}\begin{pmatrix} I^{(k)} & D^{(k)} \\ I^{(k)} & -D^{(k)} \end{pmatrix} &= \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\langle 0| \otimes I^{(k)} \\ &\quad + \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)\langle 1| \otimes D^{(k)} \\ &= (H|0\rangle\langle 0|) \otimes I^{(k)} + (H|1\rangle\langle 1|) \otimes D^{(k)} \\ &= (H \otimes I^{(k)})(|0\rangle\langle 0| \otimes I^{(k)} + |1\rangle\langle 1| \otimes D^{(k)}). \end{aligned}$$

Quantum Implementation (Cont'd)

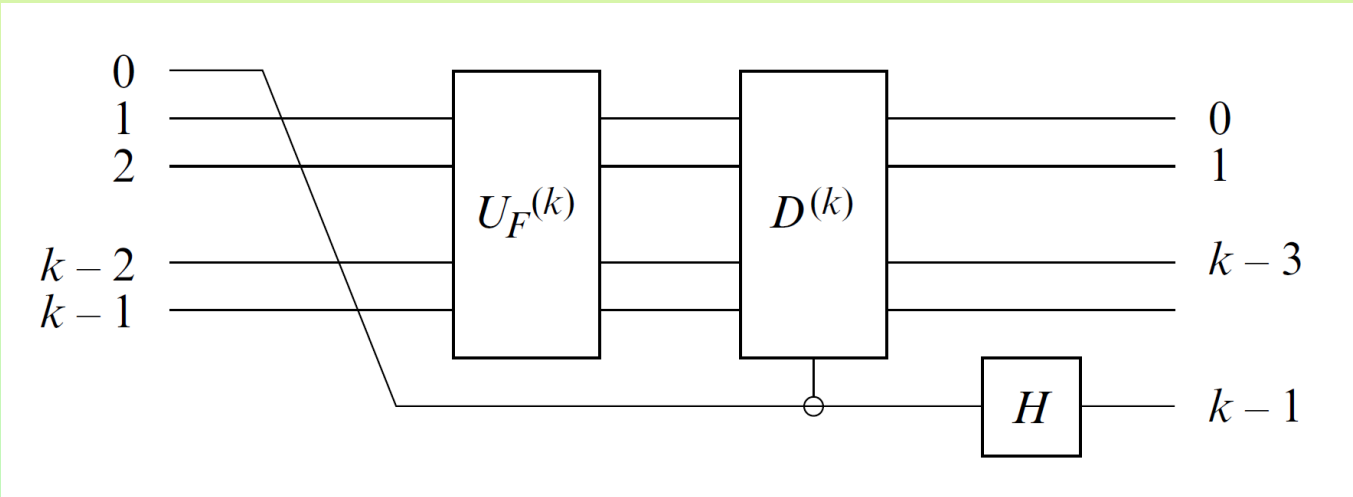
- Consider the transformation

$$(|0\rangle\langle 0| \otimes I^{(k)} + |1\rangle\langle 1| \otimes D^{(k)}).$$

- It applies $D^{(k)}$ to bits 0 through $k-1$, if bit k is one.
- That is, it applies $D^{(k)}$ to the low-order bits controlled by the high-order bit.
- This controlled version of $D^{(k)}$ can be implemented as a sequence of k two-qubit controlled gates.
- Those apply each of the single-qubit operations making up $D^{(k)}$ to bit i controlled by bit k.

Quantum Implementation (Summary)

- $D^{(k)}$ and $R^{(k)}$ can be implemented with $O(k)$ operations.
- So the kth step in the recursion adds $O(k)$ steps to the implementation of $U_F^{(n)}$.
- Overall, $U_F^{(n)}$ takes $O(n^2)$ gates to implement.
- This is exponentially faster than the $O(n2^n)$ steps required for classical fast Fourier transform.
- Circuit for the implementation of the quantum Fourier transform:



The Recursive Program

- A recursive program for the implementation is as follows.

define $QFT\ |x[1]\rangle = H|x\rangle$
$QFT\ |x[n]\rangle =$
$Swap\ |x_0\rangle|x_1 \ldots x_{n-1}\rangle$
$QFT\ |x_0 \ldots x_{n-2}\rangle$
$|x_{n-1}\rangle$ **control** $D^{(n-1)}|x_0 \ldots x_{n-2}\rangle$
$H|x_{n-1}\rangle$.