Quantum Computing and Shor's Algorithm

RAGUPATHI

MAY, 2022

Abstract

This paper covers the basic aspects of the quantum mechanics in general and quantum computing in particular, underscoring the differences between quantum computing and classical computing. This paper culminates in a discussion of Peter Shor's algorithm, a quantum computational algorithm for factoring numbers that runs in polynomial time, making it faster than any known classical algorithm for factorization. This paper serves as a survey of Polynomial Time Algorithms for Factorization on a Quantum Computer by Peter Shor.

Contents

1 Introduction

With the development of computability theory, many important problems in computer science and mathematics exist for which there is no known polynomial-time algorithm. The physicist Richard Feynman seems to have been the first to observe that quantum mechanics might allow for faster computation than Would be allowed on a classical Turing machine, and thus raised the possibility that a quantum mechanical computer might allow for more robust solutions to problems that have long plagued classical computer science. Among the problems for which no known polynomial-time deterministic algorithm exists is the integer factorization problem, which is thought to be difficult enough to compute that it forms the basis of most of modern cytological systems. Shor's algorithm, which this paper provides an exploration of, provides an efficient polynomial time algorithm that operates on a quantum computer. Although it has not been formally proven, this has led to speculation that the class of problems effectively computable by a quantum Turing machine might be broader than the class of problems computable on classical Turing machines.

2 Basics and Definitions

Here we will state results from the mathematical formulation of quantum mechanics and various aspects of quantum computing, as well as various results from other branches of mathematics as necessary. Many will be given without proof, and these terms will be referred to throughout the paper. The reader may skip ahead to the main discussion and refer back to this section as necessary.

Definition 1 (Hilbert Space)**.** *A Hilbert space H is a complex vector space equipped with an inner product $\langle x, y\rangle$ that assigns a complex number c for every pair $x, y \in H$, which satisfies the following properties:*

1. *The inner product of a pair of elements is the complex conjugate of the reverse:*

$$\langle y, x\rangle = \overline{\langle x, y\rangle}$$

For our purposes, Hilbert Spaces will be used to define the *state space of* our system. The state space of a quantum system has dimensionality equal to the number of possible outcomes of a measurement of a given property of the system. The various outcomes are each represented as vectors that they form a normalized, orthogonal basis for the state space. The inner product in the space is typically defined in terms of the basis vectors b_i of the system, such that:

Each of these b_i denote a possible outcome of measurement, with the set of all b_it's corresponding to the set of all possible measurements of a system. Thus, the inner product as defined above can be stated informally as 'selecting' the component of a state corresponding to a given outcome.

Definition 2 (Qubit)**.** *A quantum bit, or qubit, is a 2-state quantum system corresponding to a bit of data on a classical computer. In quantum computing, a qubit is represented as a linear superposition of 1's and 0's, expressed in kept notation, i.e.* 0 *and For systems of multiple qubits, it is often customary to collapse the state into multi-qubit states, as in* 010 *, which corresponds to a 0 on the first qubit, a 1 on the second, and so on. Therefore, our computational basis has* 2^n *basis states, corresponding to every possible measurement of each qubit.*

Definition 3 (Quantum Entanglement)**.** *A quantum system is said to be entangled when pairs or groups of particles are generated such that they cannot be entirely described independently, rather they must be described together as aninteracting object to fully describe the system.*

Definition 4 (BQP**).** *A decision problem is said to be in class* BQP *if there exists a quantum algorithm that solves the problem with a uniform polynomial-size quantum gate array such that the probability of returning an incorrect answer is*

At most 1/3*. Note: the probability bound is arbitrary, for any desired accuracy can be achieved by running the algorithm a polynomial number of times and taking the majority vote. This is the definition we will take for efficiently solvable problems on a quantum computer.*

3 Quantum Computing

In this section we will give an overview of quantum computing, and in particular the components necessary to understand Shor's algorithm. To begin, considera system of n two-state components. Normally, the object would be completely describable with n bits. For a quantum computer, however, this system requires 2^n 1 complex numbers, specifically this state is represented as a single point in a 2^n dimensional vector space. For each of the 2^n permutations of the system, there exists a basis vector in the state space for our system, such that the setof all 2^n of these vectors form a complete basis for the system. As an example, the state vector 000 . . . 0 would correspond to the state such that every bit is given the value 0, while the state vector 10101 . . . corresponds to the state with the first qubit in the 1 state, the second in the 0 state, and so on. We then.

Consider the state $|\psi\rangle$ of the system to be the sum of each state vector $|S_i\rangle$ with an associated complex number a_i.

$$\psi = \sum a_i |S_i\rangle$$

If the machine were to be measured at any point, then the state collapses into a single Eigen state of the observable, with the probability of any given $|S_i\rangle$ being selected is equal to $|a_i|^2$. After measurement, the quantum state of the system will be precisely the S_i that was returned by the measurement, thus destroying information of previous state. Therefore, when carrying out our quantum computing algorithm, we must be careful to never measure the state until after all computations have been made.

To perform operations on the quantum state in question, unitary operators will be used to implement logic gates. We will define one such gate here that will recur in our discussion of Shor's algorithm. The Hadamard Gate acts on a single in matrix form:

For our system, we must add two restrictions to make our gates of a uniform complexity class. The first is that the design of the gate array must be decidable in polynomial time, as per a classical algorithm. The second is that each number used in each matrix must be actually computable. Specifically, the first log(n) digits for a given entry must be computable in polynomial time. Both of these requirements are implemented to ensure that non-computable information is not hidden in the design of an efficient circuit that can therefore not be compared to a Turing Machine.

4 Shor's algorithm

Currently, the best classical algorithm for factoring large numbers is the number field sieve, which is $O(\exp(\log(n)^{1/3} \log \log(n)^{2/3}))$ [3]. Here we introduce Shor's algorithm, which will factor an integer in O $((\log(n))^{2} \log\log(n) \log\log\log(n))$. What will be presented here will not actually be an algorithm for integer factorization, but instead will be an algorithm for finding the order of an element $x$in the multiplicative group, in other words finding the least r such that $x^r = 1 \bmod (n)$. It is known that factorization can be reduced to finding the orderof the element, and thus this algorithm is sufficient for a factorizing algorithm with only perhaps some polynomial-time classical processing, as described in [3]. Roughly speaking, factoring an odd integer n, given an algorithm for computing the order of a random integer x mod (n), can be done by finding it order r, and computing gcd($x^{r/2}$ 1, n) using the Euclidean algorithm, whichis in polynomial time. Since ($x^{r/2}$ 1)($x^{r/2}$ + 1) = x^r 1 0 mod (n), the gcd($x^{r/2}$ 1, n) will only be a trivial divisor if r is odd or $x^{r/2}$ 1 mod (n). We will now present a brief sketch of a proof that, when applied to random x mod (n), this procedure will produce a factor of n with a probability of at least, where n has N distinct prime factors. Consider n = p, where least common multiple of the r_i's. Now consider the largest power of 2 dividing each r_i. This algorithm fails if and only if these all agree. Note that if they are all 1, then r is clearly odd and $r/2$ is not an integer. If they are all equal and greater than 1, then $x^{r/2}$ 1 mod (n) because $x^{r/2}$ 1 mod (p^{ai}) for each i. Then, by the Chinese Remainder Theorem, choosing an x mod (n) at random is the same as choosing an x_i mod (p^{ai}) for all i, where p^{ai} is, as before, the ith prime factor of n raised to its respective power. The multiplicative group is cyclic for any odd prime power p^{α}, so for any odd prime power p^{ai} the most $\frac{1}{\ }$. Because each of these represents a failed selection, this implies that the chance that we select an appropriate x is at least 1 $\frac{1}{\ }$. This works as long as n is odd and not a power of a prime, but mercifully there already exist algorithms that work classically to factor prime powers.

We now present a general proof for finding the order of x mod (n) on a quantum computer. Our computer will consist of two registers each containing integers represented in binary, as well as some workspace that will be reset to at the end of each subroutine.

Suppose we are given some x and n. To compute the least r such that x^r mod (n), first find the q such that n^2 q $2n^2$ and that q is some integer power of 2. Note that from now on q, x, and n are constants and will not be modified for the duration of the algorithm.

5 Quantum complexity classes

Currently, the relation between quantum complexity classes and their classical analogues is unknown. The practicality of a quantum computer stems largely from its ability to efficiently solve problems that are not thought to be solvable efficiently with a classical computer. If quantum computation cannot extend the class of functions that can be solved efficiently, then their use will most likely be largely relegated to mostly specialized and limited usage.
To begin our discussion of quantum complexity classes, we will begin with a brief overview of classical complexity classes. We make the following definitions in this section:

1. A problem is in P if it can be solved by a deterministic polynomial-time Turing Machine. These are the problems that are considered to be 'easy enough' to compute efficiently on a Turing Machine. An important result of complexity theory is that primarily testing is in P.

2. A problem is in NP if it is solvable by a nondeterministic polynomial-time Turing machine. A nondeterministic polynomial-time Turing machine is allowed to have any number of paths to reach an answer, but at least one of these paths must result in a Turing Machine that accepts the input and terminates in polynomial time if the answer is yes, and every path

3. Must reject if the answer is no. We define a problem *x* to be in class NP-COMPLETE NP if, given any problem *p* NP, *p* can be reducedto *x* with polynomial time and space modifications. In other words, an algorithm that can solve *x* in polynomial time can be used to show that all of NP can be solved in polynomial time.

4. Problem *x* has a *complement x* c̄reated by reversing the yes and no answers. For instance, consider the problem of primarily testing. Then the complement is determining if a number is composite.

The integer factorization problem is interesting in that there is no known polynomial-time solution, and yet it is not believed to exist in either NP-COMPLETE or in co-NP-COMPLETE. Because it is known to be NP and co-NP, if it were to exist in either class this would imply NP = co-NP, which is evidence to suggest that is not in either class. However, no efficient polynomial-time algorithm has been found. This leads some to suggest that the factorization problem instead lives in NP-intermediate, a class of problems not thought to be in NP-I (the I stands for intermediate), a hypothetical class of problems that are not in P or NP-COMPLETE, which is non-empty if and only if P = NP. It would seem that the usefulness of quantum mechanical processing would extend naturally to at least a subset of NP-I, if it were to exist. Note that integer factorization is not the only problem in BQP for which there is no known polynomial algorithm: the discrete logarithm is also shown in [3] to be in BQP, which has also been proposed to exist in NP-I. As the study of quantum computing is relatively young, even in comparison to complexity theory in general, it remains to be seen how far BQP can be extended. Ultimately, for BQP to see practicality in real-world computing problems, BQP would likely have to be extended to NP-COMPLETE in order to be efficient for most computationally expensive and interesting problems.

6 References

1. D. Coppersmith, An approximate Fourier transform useful in quantum factoring, IBM Research Report RC 19642 (1994).
2. P. Shor, Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer,SIAM J.Sci.Statist.Comput. 26 (1997) 1484