**Quantum Algorithm Implementations for Beginners**

arXiv:1804.03719v1 [cs.ET] 10 Apr 2018

1. **SHOR’S ALGORITHM FOR INTEGER FACTORIZATION**
   1. **Problem definition and background**

The integer factorization problem asks, given an integer *N* as an input, to find integers 1 *< N*1*, N*2 *< N* such that

*N* = *N*1*N*2. This problem is hardest when *N*1 and *N*2 are primes with roughly the same number of bits. If *n* denotes

the number of bits of *N* , no algor√ithm with polynomial in *n* time complexity is known. The straightforward algorithm

that tries all factors from 2 to *N* takes time polynomial in *N* , but exponential in *n*. The most efficient known

algorithm has running time *O* .exp ..3 64 *n*(log *n*)2ΣΣ [[82](#_bookmark301)]. In practice, integers with 1000 or more bits are impossible

9

to factor using known algorithms and classical hardware. The difficulty of factoring big numbers is the basis for the security of the RSA cryptosystem [[87],](#_bookmark306) one of the most widely used public-key cryptosystems.

One of the most celebrated results in quantum computing is the development of a quantum algorithm for factorization that works in time polynomial in *n*. This algorithm, due to Peter Shor and known as Shor’s algorithm [[93](#_bookmark312)], runs in *O*(*n*3 log *n*) time and uses *O*(*n*2 log *n* log log *n*) gates. The first implementation of that algorithm on a quantum computer was reported in 2001, when the number 15 was factored [[106](#_bookmark324)]. The largest integer factored by Shor’s algorithm so far is 21 [[71].](#_bookmark290)

In this section we describe Shor’s algorithm and its implementation on an IBM Q experience quantum computer with 5 qubits.

* 1. **Algorithm description**

1. *Reducing factorization to period finding* One way to factor an integer is by using modular exponentiation. Specifically, let an odd integer *N* = *N*1*N*2 be given, where 1 *< N*1*, N*2 *< N* . Pick any integer *k < N* such that gcd(*k, N* ) = 1, where gcd denotes the greatest common divisor. One can show that there exists an exponent *p >* 0 such that *kp* 1 (mod *N* ). Recall that, by definition, *x y* (mod *m*) if and only if *m* divides *x y*. Assume that *p* is the smallest such number. If we find such *p* and *p* is even, then, by the definition of the modulo operation, *N* divides

≡ ≡ −

*kp* − 1 = (*kp/*2 − 1)(*kp/*2 + 1)*.*

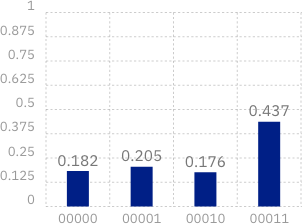
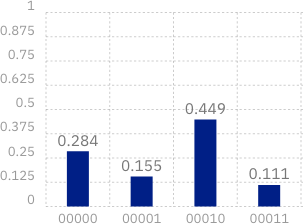
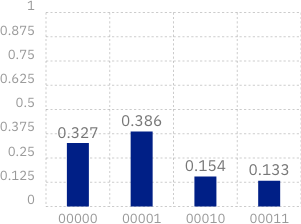


FIG. 5: Results from running the BV algorithm for 8192 shots on 2-bit hidden-strings “01”, “10” and “11” respectively (left to right) on ibmqx4. Results for runs on ibmqx5 were similar, but more noisy due to longer scores.

But since the difference between *n*1 = *kp/*2 + 1 and *n*2 = *kp/*2 1 is 2, *n*1 and *n*2 have no common factor greater than

−

2. Moreover, both numbers are nonzeros by the minimality of *p*. Since *N* = *N*1*N*2 was assumed to be odd, then *N*1 is a factor of either *n*1 or *n*2. Assume *N*1 is a factor of *n*1. Since *N*1 is also a factor of *N* , then *N*1 divides both *n*1 and *N* and one can find *N*1 by computing gcd(*n*1*, N* ). Hence, if one can compute such *p*, one can find the factors of *N* .

In order to find *p*, consider the modular exponentiation sequence *A* = *a*0*, a*1*, . . .* , where *ai* = *ki* (mod *N* ). Each *ai* is a number from the finite set 0*, . . . , N* 1 , and hence there exists indices *q* and *r* such that *aq* = *ar*. If *q* and *r* are the smallest such indices, one can show that *q* = 0 and *A* is periodic with period *r*. For instance, for *N* = 15 and *k* = 7, the modular exponentiation sequence is 1*,* 7*,* 4*,* 13*,* 1*,* 7*,* 4*,* 13*,* 1*, . . .* with period 4. Since the period 4 is an even number, we can apply the above idea to find

{ − }

74 mod 15 ≡ 1 ⇒ 74 − 1 mod 15 ≡ 0 ⇒ (72 − 1)(72 + 1) mod 15 ≡ 0 mod 15 ⇒ 15 divides 48 · 50*,*

which can be used to compute the factors of 15 as gcd(48*,* 15) = 3 and gcd(50*,* 15) = 5.

Finding the period of the sequenc√e *A* is, however, not classically easier than directly searching for factors of *N* , since

one may need to check as many as *N* different values of *A* before encountering a repetition. However, with quantum

computing, the period can be found in polynomial time using the Quantum Fourier Transform (QFT). In the next subsection we will give a definition and describe a circuit for computing QFT.

1. *Quantum Fourier Transform* The Discrete Fourier Transform (DFT) takes as an input a vector *X* of size *N*

and outputs vector *Y* = *WX* where the *Fourier matrix W* is defined by

1 1 1 *. . .* 1 

1 *ω ω*2 *. . . ωN*−1

 

√

.

.

. .

.

.

*W* =

*N* .

*,*



1

1 *ω*2 *ω*4 *. . . ω*2(*N* −1) 

1 *ωN* −1 *ω*2(*N* −1) *. . . ω*(*N* −1)(*N* −1) 

where the *ij*-the element of the matrix is *Wij* = *ωij* and *ω* is a primitive *N* -th root of unity (*ωN* = 1). A straightforward implementation of the matrix-vector multiplication takes *O*(*N* 2) operations, but, by using the special structure of the matrix, the Fast Fourier Transform (FFT) does the multiplication in only *O*(*N* log *N* ) time. The algorithm is recursive and is illustrated on Figure [6.](#_bookmark31) The Quantum Fourier Transform (QFT) is defined as a transformation between two

*yj*

*x*0

.

*xN*−2

*x*1

*x*.2 FFT

*N/*2

+

.

*xN*−1

*x*.3 FFT

*N/*2

*j*

-

*yj*+*N/*2

FIG. 6: Fast Fourier Transform circuit, where *j* denoted a row from the top half of the circuit and *ωj* denotes that the corresponding value is multiplied by *ωj*. The plus and minus symbols indicate that the corresponding values have to be added or subtracted, respectively.

quantum states that are determined using the values of DFT (FFT). If *W* is a Fourier matrix and *X* = {*xi*} and

*Y* = {*yi*} are vectors such that *Y* = *WX*, then QFT is defined as the transformation

*N* −1 *N* −1

QFT ( Σ *xk* |*k*)) = Σ *yk* |*k*) *.*

*k*=0

*k*=0

The implementation of QFT mimics the stages (recursive calls) of FFT, but implements each stage using only *n* + 1 additional gates per stage. A single Hadamard gate on the last (least significant) bit implements the additions/sub- tractions of the outputs from the recursive call and the multiplications by *ωj* are done using *n* controlled phase gates. The circuit for *n* = 5 is shown on Figure [7.](#_bookmark32)



1

ℋ

π



4

π



8

π



16

2

ℋ

π



2

π



4

π



8

3

ℋ

π



2

π



4

4

ℋ

π



2

5

ℋ

π



2

FIG. 7: A Quantum Fourier Transform circuit for five qubits (*n* = 5).

The property of QFT that is essential for the factorization algorithm is that it can “compute” the period of a periodic input. Specifically, if the input vector *X* is of length *M* and period *r*, where *r* divides *M* , and its elements are of the form

.√

*x* = *r/M* if *i* mod *r* ≡ *s*

*i* 0 otherwise

*i*=0

for some offset *s < r*, and QFT .Σ*M*

*i*=0

*xi* |*i*)Σ = Σ*M yi* |*i*), then

*y* = 1*/*√*r* if *i* mod *M/r* ≡ 0

*i* 0 otherwise

i.e., the output has nonzero values at multiples of *M/r* (the values *r/M* and 1*/*√*r* are used for normalization). Then,

.

√

in order to factor an integer, one can find the period of the corresponding modular exponentiation sequence using QFT, if one is able to encode its period in the amplitudes of a quantum state (the input to QFT).

A period-finding circuit for solving the integer factorization problem is shown on Figure [8](#_bookmark33) [[29](#_bookmark249)]. The first QFT on

register 𝐴

𝑚 qubits

0

measure

FIG. 8: Illustration of the period-finding circuit, where *m* = 2*n* and *M* = 2*m*.



QFT*M*

*f*(*i*) =

*xi* mod *N*

QFT*M*

register 𝐵

𝑛 qubits

0

register *A* produces an equal superposition of the qubits from *A*, i.e., the resulting state is

*M*

Σ 1

√

*M*

*i*=0

|*i,* 0) *.*

Next is a modular exponentiation circuit that computes the function *f* (*i*) = *xi* (mod *N* ) on the second register. The resulting state is

*M*

Σ 1

√

*M*

*i*=0

|*i, f* (*i*)) *.*

Before we apply the next QFT transform, we do a measurement of register *B*. (By the principle of deferred measurement [[75](#_bookmark294)] and due to the fact that register *A* and *B* don’t interact from that point on, we don’t have to

actually implement the measurement, but it will help to understand the final output.) If the value measured is *s*, then the resulting state becomes

1

√*M/r*

*M*

*i*=0 *f* (*i*)=*s*

Σ

|*i, s*) *,*

where *r* is the period of *f* (*i*). In particular, register *A* is a superposition with equal non-zero amplitudes only of *i* for which *f* (*i*) = *s*, i.e., it is a periodic superposition with period *r*. Given the property of QFT, the result of the transformation is the state

| )

*r*

Σ 1

√*r*

*i*=0

|*i*(*M/r*)*, s*) *.*

Hence, the measurement of register *A* will output a multiple of *M/r*. If the simplifying assumption that *r* divides *M* is not made, then the circuit is the same, but the classical postprocessing is a bit more involved.

* 1. **Algorithm implemented on IBM’s 5-qubit computer**

We implemented the algorithm on ibmqx4, a 5-qubit quantum processor from the IBM Quantum Experience, in order to factor number 15 with *x* = 11. The circuit as described on Figure [8](#_bookmark33) requires 12 qubits and 196 gates, too large to be implemented on ibmqx4. Hence, we used an optimized/compiled version from [[106](#_bookmark324)] that uses 5 qubit and 11 gates (Figure [9).](#_bookmark35) The results from the measurements are shown on Figure [10](#_bookmark36).



q0

0)

V

q1

0)

V

V

q2

Π

Π

0) V 9 9 V

4 2

q3

0)

q4

0)

V

Π

9

2

FIG. 9: Circuit for Shor’s algorithm for *N* = 15 and *x* = 11.

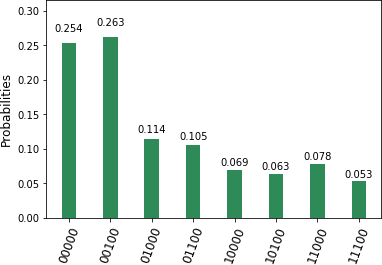
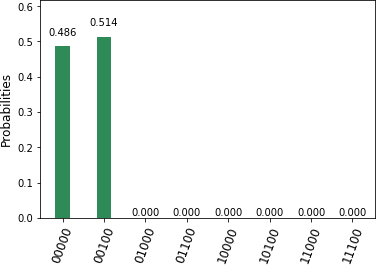


FIG. 10: Output from the circuit from Figure [9](#_bookmark35) implemented on the simulator (left) and ibmqx4 (right).

The periods found by the simulator are *p* = 0, which is ignored as a trivial period, and *p* = 4, which is a good one.

Since *M* = 8, we can conclude that *r* divides *M/p* = 8*/*4 = 2, hence *r* = 2. Then 15 divides

(*xr* − 1) = (112 − 1) = (11 − 1)(11 + 1) = 10 · 12*.*

By computing gcd(15*,* 10) = 5 and gcd(15*,* 12) = 3, we find the factors of 15.

The output from ibmqx4 finds the same periods 0 and 4 with the highest probabilities, but contains much more noise.

* 1. **Conclusion**

Shor’s quantum factorization algorithm reduces to finding the period of a periodic sequence, and such period can be found using Quantum Fourier Transform. Unless optimized, the general-case circuit for Shor’s algorithm for factorization of the number 15 is too large, both with respect to the number of qubits as well as the number of gates, to be implemented on a 5-qubit processor. We were able to implement a compiled optimized version that produced correct results, however, comparing the results produced by the simulator and the real quantum processor showed the considerable noise of the latter.