CSCI 4140
Shor’s Algorithm

Mark Green

Faculty of Science

Ontario Tech

Introduction

- Seen that Shor's algorithm is very important in quantum computing
- Theoretically the algorithm is quite simple, but practically it is very difficult to implement efficiently
- Quickly review Shor's algorithm
- Highlight the problem with implementation
- Do a simplified version of the problem in Qiskit

Shor’s Algorithm

1. If N is even, return 2
2. If $N = p^q$, return p. This can be done in polynomial time on a classical computer
3. Choose a random number a such that $1 < a \leq N - 1$, if gcd(a,N)>1 return gcd(a,N). This can be done on a classical computer
4. Use the order finding quantum algorithm to find the order r of a modulo N
5. If r is odd or $a^{\frac{r}{2}} = -1(mod\ N)$ go back to step 3. Otherwise compute $\gcd\left(a^{\frac{r}{2}} + 1, N\right)\ and\ \gcd(a^{\frac{r}{2}} - 1, N)$, if either is a non-trivial factor of N, return it. Otherwise back to step 3

Shor’s Algorithm

- The quantum part of the algorithm is in step 4, the order or period finding problem, two names for the same problem
- Consider the following function:

$$f(x) = a^x mod\ N$$

- Where:
  - $a < N$
  - a and N have no common factors
- The period of f(x) is the smallest non-zero integer r, such that

$$a^r mod\ N = 1$$

Shor’s Algorithm

- Shor’s algorithm is based on using quantum phase estimation on the following unitary operator:

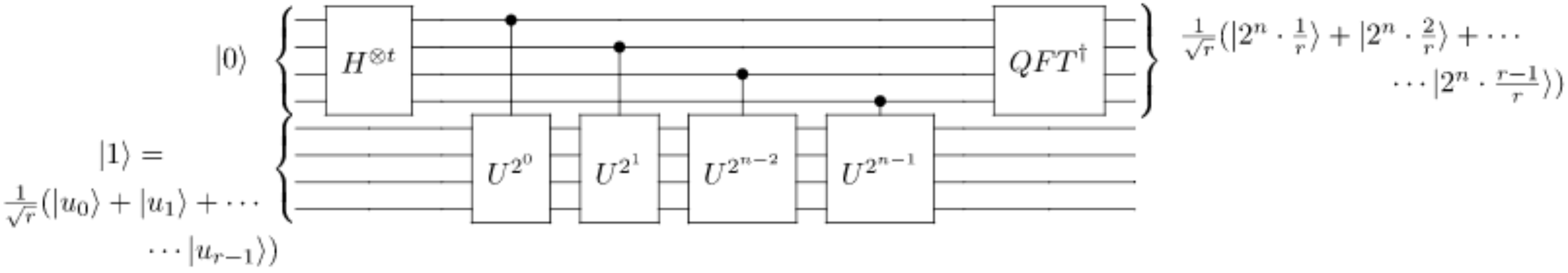
$$U|y\rangle = |ay \; mod \; N\rangle$$

- For our example function this operates in the following way:

$$U|1\rangle = |3\rangle$$
$$U^2|1\rangle = |9\rangle$$
$$\ldots$$
$$U^r|1\rangle = |1\rangle$$

Shor’s Algorithm

- Using the bits and pieces that we already have we can construct the following circuit



- The real problem here is computing U^{2^i} where U is given on the previous slide

Shor’s Algorithm

- It’s difficult to do arithmetic on a quantum computer, modular arithmetic is even harder
- There are efficient ways of computing U^{2^i} but they are quite complicated, and we won’t examine them
- It is still a research problem to do this efficiently
- Instead we will examine a restricted version of this algorithm where the circuit can be easily implemented in Qiskit
- This will give you a flavour of the algorithm

Example

- As an example we will compute the prime factors of 15
- Why 15?
  - Uses a small number of qubits
  - Everyone else does it
  - The circuit is relatively simple
- We will need our qft_dagger() function that we’ve used before
- The next slide shows the function that computes (a mod 15) to any power

Example

```python
def c_amod15(a, power):
    """Controlled multiplication by a mod 15"""
    if a not in [2,7,8,11,13]:
        raise ValueError("'a' must be 2,7,8,11 or 13")
    U = QuantumCircuit(4)
    for iteration in range(power):
        if a in [2,13]:
            U.swap(0,1)
            U.swap(1,2)
            U.swap(2,3)
        if a in [7,8]:
            U.swap(2,3)
            U.swap(1,2)
            U.swap(0,1)
        if a == 11:
            U.swap(1,3)
            U.swap(0,2)
        if a in [7,11,13]:
            for q in range(4):
                U.x(q)
    U = U.to_gate()
    U.name = "%i^%i mod 15" % (a, power)
    c_U = U.control()
    return c_U
```

Example

- Next we need a function that will determine the phase of a mod 15
- This is just our phase estimation function, changed to use our mod function
- The following slide shows what happen when we use a=8 with the above code
- Normally we would use a random number, but I want to illustrate what happens when phase estimate produces a useless result
- In this case it tells me that that the factors of 15 are 1 and 15, not very helpful

Example

```
def qpe_amod15(a):
    n_count = 3
    qc = QuantumCircuit(4+n_count, n_count)
    for q in range(n_count):
        qc.h(q)     # Initialise counting qubits in state |+>
    qc.x(3+n_count) # And ancilla register in state |1>
    for q in range(n_count): # Do controlled-U operations
        qc.append(c_amod15(a, 2**q), 
                 [q] + [i+n_count for i in range(4)])
    qc.append(qft_dagger(n_count), range(n_count)) # Do inverse-QFT
    qc.measure(range(n_count), range(n_count))
    # Simulate Results
    backend = Aer.get_backend('qasm_simulator')
    # Setting memory=True below allows us to see a list of each sequential reading
    result = execute(qc, backend, shots=1, memory=True).result()
    readings = result.get_memory()
    print("Register Reading: " + readings[0])
    phase = int(readings[0],2)/(2**n_count)
    print("Corresponding Phase: %f" % phase)
    return phase
```

Example

```
a=8
phase = qpe_amod15(a)
frac = Fraction(phase).limit_denominator(15)
s, r = frac.numerator, frac.denominator
print(r)
guesses = [gcd(a**(r//2)-1, N), gcd(a**(r//2)+1, N)]
print(guesses)
```

```
Register Reading: 000
Corresponding Phase: 0.000000
1
[15, 1]
```

Example

- There is a good chance that our phase estimation algorithm will find the trivial factors
- To avoid this problem we need to run our code multiple times
- Since we are dealing with a probabilistic result, the phase estimation algorithm can return a different phase each time we run it
- We need to check this phase to see if it produces non-trivial factors
- The complete code for the quantum part of Shor’s algorithm is shown on the next slide

Example

```python
N=15
np.random.seed(2) # This is to make sure we get reproduceable results
a = randint(2, N)
while gcd(a,N) != 1:
    a = randint(2,15)
print("a: %i"%a);

factor_found = False
attempt = 0
while not factor_found:
    attempt += 1
    print("\nAttempt %i:" % attempt)
    phase = qpe_amod15(a) # Phase = s/r
    frac = Fraction(phase).limit_denominator(15) # Denominator should (hopefully!) tell us r
    r = frac.denominator
    print("Result: r = %i" % r)
    if phase != 0:
        # Guesses for factors are gcd(x^{r/2} ±1 , 15)
        guesses = [gcd(a**(r//2)-1, 15), gcd(a**(r//2)+1, 15)]
        print("Guessed Factors: %i and %i" % (guesses[0], guesses[1]))
        for guess in guesses:
            if guess != 1 and (15 % guess) == 0: # Check to see if guess is a factor
                print("*** Non-trivial factor found: %i ***" % guess)
                factor_found = True
```

Example

- If we run this code we get the following result
- Note that it takes two tries to find non-trivial factors

```
a: 8

Attempt 1:
Register Reading: 000
Corresponding Phase: 0.000000
Result: r = 1

Attempt 2:
Register Reading: 100
Corresponding Phase: 0.500000
Result: r = 2
Guessed Factors: 1 and 3
*** Non-trivial factor found: 3 ***
```

Summary

- Examined how Shor's algorithm can be implemented in Qiskit
- Seen that the real problem is evaluating U^{2^i} when U is (a mod N)
- Doing this efficiently is a hard problem
- Instead used N=15, where the circuit is relatively easy to produce