Implementation of the discrete log problem on IBM's quantum computing toolkit

1st Dhruvesh Asnani
201601423 DA-IICT
Gandhinagar, Gujarat, India
201601423@daiict.ac.in

***Abstract*—In our B.Tech Project, we study the discrete log problem (DLP) which is a a special case of the hidden subgroup problem (HSP). In particular, we study the DLP over $\mathbb{Z}_N^*$, the multiplicative group of integers modulo N. We give a complete implementation of a polynomial time quantum algorithm for DLP over $\mathbb{Z}_N^*$ on IBM's quantum computing platform QisKit.**

I. Introduction

Since the discovery of Shor's algorithm [5] for order finding, the search for polynomial time quantum algorithms for other problems, which were believed to be classically hard, began extensively. It turned out that many problems, for which efficient quantum algorithms existed, were special cases of the Hidden Subgroup Problem (HSP) [6]. The discrete log problem is one of them. Our study is motivated by the fact that an efficient algorithm for discrete log will break many important classical cryptographic protocols (for e.g. Diffie-Hellman key exchange).

As a part of our B.Tech project, we studied quantum algorithms for 3 special cases of the HSP for finite groups: Abelian HSP, Non-Abelian HSP where the hidden subgroup is normal and Kuperberg's algorithm for Dihedral HSP where the order of the dihedral group is 2^n ($n \in \mathbb{Z}^+$). Finally, we implemented a polynomial time quantum algorithm for the discrete log problem over $\mathbb{Z}_N^*$ ($N \in \mathbb{Z}^{\geq 2}$) using IBM QisKit.

This report assumes the knowledge of fundamentals of quantum computation and of group theory.

II. The Hidden Subgroup Problem and some motivation

Let G be a group and S be a set. Let $H \leq G$. We say that a function $f : G \to S$ hides H in G iff

$$\text{for all } x, y \in G \; f(x) = f(y) \text{ if and only if } xy^{-1} \in H$$

In simple words, two elements of G map to the same element of S if and only if they are in the same right coset of H in G. One can also give a definition in terms of left cosets. We can use any definition as long as we use it consistently. Here is the formal statement of the Hidden Subgroup Problem:

Given a function $f : G \to S$ that hides a subgroup H in G, find (a generating set for) H.

If G is an abelian group, then the problem is called Abelian HSP. If G is the dihedral group then the problem is called Dihedral HSP. We will only consider the case where G is finite. So, let G be a finite group.

First look suggests that this is an abstract algebraic problem. Now, we will give some motivation as to why we should study this problem.

Now, we define the discrete log problem (DLP) over a finite group G'. Let $a, b \in G'$ such that $b = a^t$. t is called a discrete logarithm of b to the base a. DLP asks: given $a, b \in G'$, find a discrete logarithm of b to the base a if one exists. Otherwise report the non-existence of such a discrete logarithm.

Now, we show that discrete log problem over G' is an instance of finite abelian HSP. Let $|a| = n$. If $n = 1$, the problem is easy to solve. Now let $n \geq 2$. WLOG, let $t \in \mathbb{Z}_n$. Let $G = \mathbb{Z}_n \times \mathbb{Z}_n$, $H = \langle (t, 1) \rangle$ and $S = G'$. Define $f : G \to S$ as

$$f(x, y) = a^x b^{-y} \tag{1}$$

Let $g_1 = (x_1, y_1)$ and $g_2 = (x_2, y_2)$ such that $g_1, g_2 \in G$.

$$\begin{aligned} & f(g_1) = f(g_2) \\ \Leftrightarrow \quad & a^{x_1} b^{-y_1} = a^{x_2} b^{-y_2} \\ \Leftrightarrow \quad & a^{x_1 - x_2} = b^{y_1 - y_2} = a^{t(y_1 - y_2)} \\ \Leftrightarrow \quad & (x_1 - x_2) \equiv t(y_1 - y_2) \bmod n \\ \Leftrightarrow \quad & g_1 - g_2 \in H \end{aligned}$$

This shows that f indeed hides H in G. So, on solving this HSP, we can find H and in particular find $(t, 1)$, which gives t. If the instance of DLP has no solution then this will be indicated by the failure of the algorithm. Thus, an efficient algorithm for finite abelian HSP implies an efficient algorithm for the discrete log problem over finite groups.

Also, the order-finding problem for a finite group, which asks to find the order of a given element of the group, can also be shown to be an instance of finite abelian HSP.

III. Quantum Algorithms for HSP

Ettinger, Høyer and Knill [7] showed that the quantum query complexity of HSP is polynomial, i.e., in quantum case, one needs to make $\mathcal{O}(\text{poly}(\log |G|))$ queries to f to find H. Here, $\text{poly}(\log |G|)$ denotes a polynomial in $\log |G|$. This result was a positive direction in search of efficient algorithms for HSP.

Weak Fourier Sampling is an quantum algorithmic technique which is most common in solving HSP. The key part in it is the Quantum Fourier Transform (QFT) over G. It can

be shown that under certain reasonable assumptions, Weak Fourier Sampling is sufficient to identify any normal hidden subgroup in $\mathcal{O}(\text{poly}(\log|G|))$ time [4]. This immediately gives an efficient algorithm for abelian HSP since every subgroup of an abelian group is normal.

If G is non-abelian and the hidden subgroup is not normal, then there is no known general algorithm. There are only algorithms specific to the type of group G. Once such algorithm we studied is Kuperberg's algorithm for Dihedral HSP [8]. This algorithm uses weak fourier sampling as a beginning step and then does more quantum processing to find the hidden subgroup. We studied this algorithm only for the case where order of the dihedral group is 2^n for some $n \in \mathbb{Z}^+$. The algorithm for the general case is much more involved.

Here, we won't discuss the details of any of the algorithms that we studied because that requires some background in representation theory and is too involved to discuss here. To see the details, refer this link.

IV. Discrete Log Problem and its implementation

For the rest of the report we will discuss about constructing a quantum circuit for solving the discrete log problem over $\mathbb{Z}_N^*$ where $N \in \mathbb{Z}^{\geq 2}$. We will not use the quantum algorithm for finite abelian HSP because that approach requires implementing QFT over $\mathbb{Z}_n$ for arbitrary $n \in \mathbb{Z}^+$. No efficient circuit to implement it exactly, for all $n \in \mathbb{Z}^+$, is known. The approach we now take requires QFT over $\mathbb{Z}_{2^n}$ ($n \in \mathbb{Z}^+$) for which there is a very efficient and easy-to-implement circuit. The implementation of the circuit for DLP is inspired from [2]. We first discuss an important concept in quantum computation which will form the basis of the algorithm.

A. Phase estimation

The phase estimation algorithm is used to determine the eigenvalue of a unitary operator U corresponding to the given eigenvector. Since the eigenvalues of a unitary operator lie on the unit circle in the complex plane, they are of the form $e^{i2\pi\phi}$ where $\phi \in [0, 1)$. So, in order to determine the eigenvalue it is sufficient (and necessary) to determine the phase $2\pi\phi$ of the eigenvalue and hence the name 'phase estimation'.

Let $|\phi\rangle$ be an eigenvector of U such that $U|\phi\rangle = e^{i2\pi\phi}|\phi\rangle$. Let $m \in \mathbb{Z}^+$ be the number of qubits required to represent $|\phi\rangle$ in a quantum register. In the figure below, $n \in \mathbb{Z}^+$ is related to the accuracy with which ϕ needs to be determined. The circuit for phase estimation can be summarized as follows:

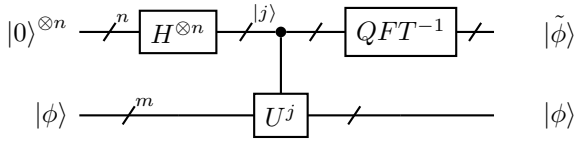

Fig. 1. Phase estimation circuit

The vector $|\tilde{\phi}\rangle$ in the figure above is called the estimator of ϕ. On measuring the first n qubits, we will obtain $j \in \mathbb{Z}_{2^n}$ such that $\frac{j}{2^n}$ is an approximation of ϕ. Specifically, with probability atleast $\frac{4}{\pi^2}(\approx 0.4)$ we will obtain a j such that $\left|\phi - \frac{j}{2^n}\right| \leq \frac{1}{2^{n+1}}$ [2]. It can also be shown that with probability atleast $\frac{8}{\pi^2}(\approx 0.81)$ we will obtain a j such that $\left|\phi - \frac{j}{2^n}\right| \leq \frac{1}{2^n}$.

B. The algorithm

The algorithm for discrete log problem is based on phase estimation. Let $a, b \in \mathbb{Z}_N^*$ ($N \in \mathbb{Z}^{\geq 2}$) such that $b = a^t \bmod N$. Let the order of a modulo N be r. r can be determined in polynomial time using Shor's algorithm. WLOG, let $t \in \mathbb{Z}_r$. If $r = 1$, then $a = 1$ and the problem has a solution if and only if $b = 1$ (in this case $t = 0$). For the remaining discussion, let $r \geq 2$.

Let $m = \lceil \log_2(N+1) \rceil$, i.e., m is the minimum number of bits needed to represent N in binary. Now, for $a \in \mathbb{Z}_N^*$, define U_a as follows:

$$U_a|x\rangle = |ax \bmod N\rangle \qquad \text{where } x \in \mathbb{Z}_{2^m} \tag{2}$$

To implement U_a as a quantum circuit, we have to first make it into a unitary operator. Doing so may require some auxiliary qubits. Here we do not show these auxiliary qubits and assume that they are all set to $|0\rangle$ throughout.

Consider the following vectors for $k \in \mathbb{Z}_r$:

$$|u_k\rangle = \frac{1}{\sqrt{r}} \sum_{j=0}^{r-1} \omega_r^{-jk} \left|a^j \bmod N\right\rangle \tag{3}$$

where $\omega_r := e^{i\frac{2\pi}{r}}$.

Each of these is an eigenvector of U_a, i.e., for $k \in \mathbb{Z}_r$,

$$U_a|u_k\rangle = \omega_r^k |u_k\rangle \tag{4}$$

Also, their normalized sum is $|1\rangle$:

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |u_k\rangle = |1\rangle \tag{5}$$

Now, since, $b = a^t \bmod N$, $U_b = U_a^t$. So, for $k \in \mathbb{Z}_r$,

$$\begin{aligned} U_b|u_k\rangle &= U_a^t|u_k\rangle \\ &= \omega_r^{kt}|u_k\rangle \\ &= \omega_r^{kt \bmod r}|u_k\rangle \end{aligned}$$

So, $|u_k\rangle$, for each $k \in \mathbb{Z}_r$, is an eigenvector of U_b. Now, consider the following phase estimation circuit:

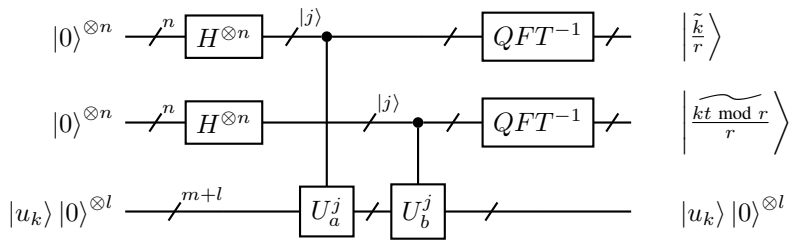

Fig. 2. Phase estimation ciruit for DLP

The above circuit can be thought of as consisting of two phase estimation circuits. Since U_a and U_b share eigenvectors, the lower parts of the phase estimation circuit can be put in

succession(i.e. in series). So, for $k \in \mathbb{Z}_r$, the above circuit implements that following transformation:

$$|0\rangle^{\otimes n} |0\rangle^{\otimes n} |u_k\rangle |0\rangle^{\otimes l} \mapsto \left|\frac{\tilde{k}}{r}\right\rangle \left|\frac{\widetilde{kt \bmod r}}{r}\right\rangle |u_k\rangle |0\rangle^{\otimes l}$$

where the last l qubits form the auxiliary register. Later, we will see that $l = m + 2$.

Now, instead of a particular eigenvector $|u_k\rangle$, if we input $|1\rangle$, then according to (5), we will get:

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} \left|\frac{\tilde{k}}{r}\right\rangle \left|\frac{\widetilde{kt \bmod r}}{r}\right\rangle |u_k\rangle |0\rangle^{\otimes l}$$

So, by measuring the first two registers, we can find $\frac{k}{r}$ and $\frac{kt \bmod r}{r}$ where each $k \in \mathbb{Z}_r$ occurs with probability $\frac{1}{r}$. Since we know r, we can find k and $kt \bmod r$. Repeat the algorithm sufficient number of times to find $k_1, k_2 \in \mathbb{Z}_r$ such that $\gcd(k_1, k_2) = 1$. Let $v_1 = k_1 t \bmod r$ and $v_2 = k_2 t \bmod r$. Since $\gcd(k_1, k_2) = 1$, there exist $\lambda_1, \lambda_2 \in \mathbb{Z}$ such that $\lambda_1 k_1 + \lambda_2 k_2 = 1$. Since $t \in \mathbb{Z}_r$,

$$\begin{aligned} t &= t \bmod r \\ &= (\lambda_1 k_1 t + \lambda_2 k_2 t) \bmod r \\ &= (\lambda_1 v_1 + \lambda_2 v_2) \bmod r \end{aligned}$$

In this way, we can find t.

C. Implementing controlled-U_a

Now, we discuss how to implement the transformation U_a as a quantum circuit. Our approach is taken entirely from [1]. Only an outline of the approach is presented here. The details are explained in a very readable manner in [1].

The circuit is built incrementally in the following stages. In the following, let a, b and N be n-bit integers ($n \in \mathbb{Z}^+$) and let $|\phi(b)\rangle$ denote QFT of $|b\rangle$. The number of bits in QFT will be clear from context. Also, a, n and N must be known prior to making the circuits. Each circuit below makes direct or indirect use of every circuit built prior to itself.

1) $\Phi ADD(a)$: This circuits adds a to b modulo 2^n. The input and output are in n-bit Fourier space.

$$|\phi(b)\rangle \mapsto |\phi((a+b) \bmod 2^n)\rangle$$

It requires $\Theta(n)$ gates.

2) $\Phi ADD(a) MOD(N)$: Let $b \in \mathbb{Z}_N$. This circuit adds a to b modulo N. The input and output are in $(n+1)$-bit Fourier space.

$$\begin{aligned} |c_1\rangle |c_2\rangle |\phi(b)\rangle |0\rangle &\mapsto |c_1\rangle |c_2\rangle |\phi((a+b) \bmod N)\rangle |0\rangle \\ &\qquad \text{if } c_1 \cdot c_2 = 1 \\ &\mapsto |c_1\rangle |c_2\rangle |\phi(b)\rangle |0\rangle \\ &\qquad \text{otherwise} \end{aligned}$$

It is doubly-controlled using c_1 and c_2. It uses 1 auxiliary qubit. It requires $\Theta(n^2)$ gates.

3) c-U_a: Let $a \in \mathbb{Z}_N^*$. This circuit multiplies a to b modulo N.

$$\begin{aligned} |c\rangle |b\rangle |0\rangle &\mapsto |c\rangle |ab \bmod N\rangle |0\rangle \qquad \text{if } c = 1 \\ &\mapsto |c\rangle |b\rangle |0\rangle \qquad \text{if } c = 0 \end{aligned}$$

c is the control bit. The rightmost register is an auxiliary quantum register of $n+2$ bits. It requires $\Theta(n^3)$ gates.

D. Analysis

Here we will analyze the success probability and the gate complexity of the phase estimation circuit for DLP.

In the phase estimation circuit, let $n = \lceil \log_2(r+1) \rceil + 1$. So, $r < 2^{n-1}$. Let $v \in \mathbb{Z}_{2^n}$ be obtained by measuring the first register. Then we know that, for given k, with probability atleast $\frac{8}{\pi^2}$, it is the case that:

$$\begin{aligned} &\left|\frac{k}{r} - \frac{v}{2^n}\right| \leq \frac{1}{2^n} \\ \Rightarrow \quad &\left|k - \frac{vr}{2^n}\right| \leq \frac{r}{2^n} < \frac{1}{2} \end{aligned}$$

This shows that for given k, with probability atleast $\frac{8}{\pi^2}$, rounding $\frac{vr}{2^n}$ to the nearest integer gives k.

The above argument also holds for the second register where we try to obtain $kt \bmod r$. Also, we want two measurements such that k_1 and k_2 are relatively prime. The probability of two integers in $\mathbb{Z}_r$ being relatively prime is atleast $\frac{1}{2}$ for any $r \in \mathbb{Z}^{\geq 2}$ (See *Appendix*). So, we will successfully obtain t with probability at least $\frac{1}{2}\left(\frac{8}{\pi^2}\right)^4 (\approx 0.22)$. We can improve the success probability by increasing n [3].

The key point in the above analysis is that the success probability is lower bounded by a constant. So, the phase estimation circuit needs to be run atmost a constant number of times on average.

The phase estimation circuit uses $\Theta(n)$ Hadamard gates. The c-U_a and c-U_b exponentiation requires $\Theta(nm^3)$ gates. The inverse QFTs require $\Theta(n^2)$ gates. So, the overall gate complexity is $\Theta(n^2 + nm^3)$. In practice, $n = \mathcal{O}(m)$ suffices. In that case, the gate complexity is $\mathcal{O}(m^4)$. Since $m = \Theta(\log N)$, the gate complexity is $\mathcal{O}(\log^4 N)$.

The $\mathcal{O}(\log^4 N)$ gate complexity along with the fact that the success probability is lower bounded by a constant gives a quantum algorithm with running time of $\mathcal{O}(\log^4 N)$ on average, for DLP over $\mathbb{Z}_N^*$. Note that here we do not take in into account the time complexity for finding r. But we know that Shor's algorithm can find r in time proportional to a polynomial in $\log N$ on average. So, the overall running time will still be polynomial in $\log N$ on average. In contrast, every known classical algorithm for DLP over $\mathbb{Z}_N^*$ has running time superpolynomial in $\log N$ [2].

V. Conclusion

We wrote an implementation of the above quantum algorithm using IBM QisKit in Python. The details about running the code are in the appendix. The code only performs a classical simulation of our quantum circuit. It does not run on an actual quantum computer. We tried to run it on one of

IBM's actual quantum computer but it turns out that even for very small values of r and N, the circuit is too deep to be run on the quantum computer. But it is only a matter of time before deep circuits can be run on actual quantum computers.

It turns out that QisKit allows one to simulate actual quantum computer-like scenario by adding noise to gates and measurements. Our code allows one to set these noise parameters and experiment.

ACKNOWLEDGMENT

We would like to thank Prof. Gautam Dutta, Prof. Aditya Tatu and Prof. Jaideep Mulherkar for their constant feedback and guidance throughout the project.

REFERENCES

[1] Stephane Beauregard (2002), Circuit for Shor's algorithm using $2n+3$ qubits, quant-ph/0205095
[2] P. Kaye, R. Laflamme, and M. Mosca (2007), An Introduction to Quantum Computing, Oxford University Press
[3] R. Cleve, A. Ekert, C. Macchiavello, and M. Mosca (1998), Quantum algorithms revisited, quant-ph/9708016
[4] A. Childs, Lecture Notes on Quantum Algorithms
[5] P. Shor (1997), Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer, quant-ph/9508027
[6] M. Nielsen, and I. Chuang (2000), Quantum Computation and Quantum Information, Cambridge University Press
[7] M. Ettinger, P. Høyer, and, E. Knill (2004), The quantum query complexity of the hidden subgroup problem is polynomial, quant-ph/0401083
[8] G. Kuperberg (2003), A subexponential-time quantum algorithm for the dihedral hidden subgroup problem, quant-ph/0302112

APPENDIX

A. Analysis of Probability

Here, we look at $p(n)$ that probability of two integers in $\mathbb{Z}_n$ ($n \in \mathbb{Z}^{\geq 2}$) being relatively prime.

Here, we will make use of the indicator function I. The input to I is a proposition p and its output $I\{p\}$ is 1 if p is true and 0 if p is false.

The indicator function makes counting arguments easier to understand. For example, the totient function can be expressed as:

$$\phi(n) = \sum_{k=1}^{n} I\{\text{gcd}(n,k) = 1\}$$

where $n \in \mathbb{Z}^+$

Now, the number of unordered pairs of the first n positive integers where the two elements are relatively prime is:

$$\begin{aligned}\Phi(n) &= \sum_{1 \leq l \leq k \leq n} I\{\text{gcd}(k,l) = 1\} \\ &= \sum_{k=1}^{n} \sum_{l=1}^{k} I\{\text{gcd}(k,l) = 1\} \\ &= \sum_{k=1}^{n} \phi(k)\end{aligned}$$

where $n \in \mathbb{Z}^+$

$\Phi(n)$ is also known as the totient summatory function.

Now, the number of ordered pairs of the first n positive integers where the two elements are relatively prime is $2\Phi(n) - 1$. 1 is subtracted because the pair $(1,1)$ is counted twice in $2\Phi(n)$.

So, the number of ordered pairs of $\mathbb{Z}_n$ where the two elements are relatively prime is $2\Phi(n-1) + 1$. Here, 2 is added to include $(0,1)$ and $(1,0)$. So,

$$p(n) = \frac{2\Phi(n-1) + 1}{n^2} \tag{6}$$

where $n \in \mathbb{Z}^{\geq 2}$

It is known that

$$\Phi(n) = \frac{3n^2}{\pi^2} + \mathcal{O}(n \log n) \tag{7}$$

See, for example, p.353, An Introduction to the Theory of Numbers (6 ed.) by Hardy and Wright.

So, for large n, $p(n) \approx \frac{6}{\pi^2}$. For small n, a plot of $p(n)$ against n for $n \geq 2$ shows that $p(n) \geq \frac{1}{2}$.

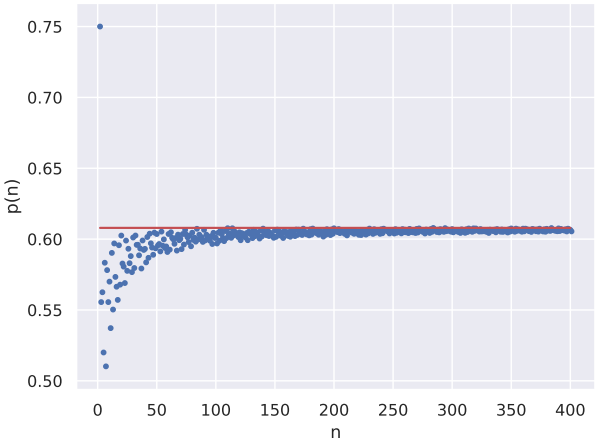

Fig. 3. $p(n)$ v/s n for $n \geq 2$

B. The code

Our code is written in a Jupyter notebook. Here is one way to run it:

1) Click on this link. Colab is a service by Google which allows to to open Jupyter notebook online without going through any installation fuss on your local PC. Also, this will require you to login into your Google account.
2) Click on 'Open in Playground' near top left of the screen.
3) Click on the first cell. Then, in the menu bar, go to Runtime → Run all.
4) Go to the last cell. Here change the values of a, b and N according to your choice. Then run only the current cell.

The code will take around 4-5 minutes to run depending upon the input and the number of simulations.