Quantum Algorithms Questions

Ayush Goyal

2022

1 1^{st} Question

Construct a reversible circuit that takes an integer between 0 and 7 as an input, and outputs the (positive) square root (rounded to the nearest integer).

1.1 Solution:

Here, we realize that integers between 0 and 7 can be input as 3-bit input circuit and would require a 2-bit output circuit. Thus we can represent it with a four-bit input $x_1x_2x_3x_4$ and two-bit square root output q_1q_2 and four-bit remainder output $r_1r_2r_3r_4$ constructed using six RCAS (Reversible Controlled Adder/Subtractor) circuits and six CNOT gates arranged in two rows to connect them.

On first consideration, we can basically use Fredkin gate and Peres gate for computation which gives XOR and remainder gates and thus, we construct the RCAS mosule using these principles.

Figure 1: (a) Fredkin gate (b) Peres gate

RCAS: This is a block to perform an addition or subtraction depending on the value of the input control signal and the implementation of an RCAS module is done using one CNOT gate and two Peres gates (or using a Haghparast quantum circuit and a CNOT gate). A one-bit QAS circuit adds three bits X,Y and Z(Carry in) and generates the sum $S(X \oplus Y \oplus Z)(or subtracts D(X \oplus Y \oplus Z \oplus 1))$ and carry-out $C_{out}((X \oplus Y)Z \oplus XY)$. Thus, the inputs being data signals X, Y, Z, a control signal A/S, and a constant input bit 0; and outputs are sum, carry, A/S_g and two garbage outputs g_1 and g_2.

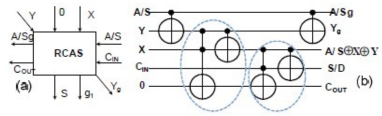

Figure 2: RCAS Module (a) block diagram (b) reversible realization

We can also construct a truth table for this RCAS module as follows:

const.	A/S	Cin	X	Y	S/D	Cout	A/S_g	Y_g	g2
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	0	0	1	1
0	0	0	1	0	1	0	0	0	1
0	0	0	1	1	0	1	0	1	0
0	0	1	0	0	1	0	0	0	0
0	0	1	0	1	0	1	0	1	1
0	0	1	1	0	0	1	0	0	1
0	0	1	1	1	1	1	0	1	0
0	1	0	0	0	1	0	1	0	1
0	1	0	0	1	0	0	1	1	0
0	1	0	1	0	0	1	1	0	0
0	1	0	1	1	1	0	1	1	1
0	1	1	0	0	0	1	1	0	1
0	1	1	0	1	1	0	1	1	0
0	1	1	1	0	1	1	1	0	0
0	1	1	1	1	0	1	1	1	1

Figure 3: Reversible controlled adder/subtractor truth table

Now, using this RCAS module , we can construct or reversible circuit consisting of two 2 rows - first with 2 RCAS and second with 4 RCAS modules. The first control input signal A/S for RCAS 1 is set to 1 and the control bits are connected to the other RCAS accordingly and CNOT gate generates a copy of the required signal for reversible implementation.

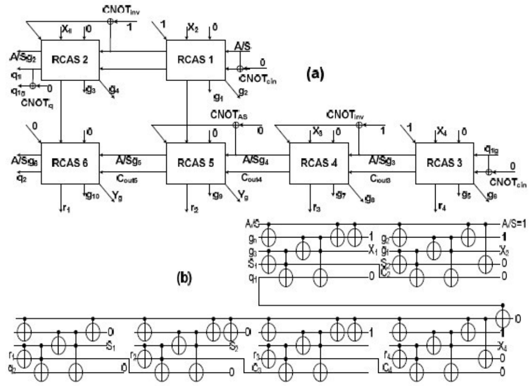

Figure 4: 4-bit reversible square root circuit (a) block diagram (b) reversible implementation

2 2^{nd} Question

Let X_1 and X_2 be random variables, and define

$$Y = a_1X_1 + a_2X_2$$

We wish to estimate the mean of Y by sampling X_1 and X_2. With q_1 samples, X_1 can be estimated with mean-squared error q_1^{-1}; and with q_2 samples, X_2 can be estimated with mean-squared error q_2^{-1}. Suppose we are allowed to take some q samples in total, (i.e. $q = q_1 + q_2$), find q_1 and q_2 such that the estimate of the mean of Y has minimum squared error.

2.1 Solution:

We understand that mean-squared value is evaluated as

$$MSE(X) : s^2 = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

and we know two other parameters of data for statistics defined as

$$Variance : \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2 \textbf{ and } Bias_\theta[\hat{\theta}] = E[\hat{\theta} - \theta]$$

Also, the relation can be described as:

$$MSE(\hat{\theta}) = Var(\hat{\theta}) + Bias(\hat{\theta}, \theta)^2$$

Here,

$$Y = a_1X_1 + a_2X_2$$

and

$$MSE(X_1) : \frac{1}{q_1} = \frac{1}{q_1}\sum_{i=1}^{q_1}(X_i - \hat{X}_i)^2$$

,

$$MSE(X_2) : \frac{1}{q_2} = \frac{1}{q_2}\sum_{i=1}^{q_2}(X_i - \hat{X}_i)^2$$

Considering all random numbers to be independent, we get the bias to be 0. Thus, we have $MSE(X) = Var(X)$.

Using this relation diectly to find MSE of Y, with $Var(Y) = a_1^2 Var(X_1) + a_2^2 Var(X_2^2) + 2abCov(X_1, X_2)$ where the $Cov(X_1, X_2) = 0$ - independent, we get

$$MSE(Y) = \frac{a_1^2}{q_1} + \frac{a_2^2}{q_2}$$

Differentiating to minimize the MSE value, we get

$$\frac{d(MSE(Y))}{dq_1} = -\frac{a_1^2}{q_1^2} + \frac{a_2^2}{(q - q_1)^2} = 0$$

(Since $q_1 + q_2 = q$)

$$\frac{a_1^2}{q_1^2} = \frac{a_2^2}{(q - q_1)^2}$$

$$\frac{q}{q_1} - 1 = \frac{a_2}{a_1}$$

$$q_2 = \frac{q}{1 + \frac{a_1}{a_2}} \textbf{ and } q_1 = \frac{q}{1 + \frac{a_2}{a_1}}$$

3 3^{rd} Question

Let some qubit, $|\psi\rangle$, be in the state:

$$|\psi\rangle = \cos\theta\,|0\rangle + \sin\theta\,|1\rangle$$

for which we define the 'amplitude', $a = \sin^2\theta$.

- If we prepare and measure some n independent copies of $|\psi\rangle$, how can we estimate a and what is the distribution of that estimator?
- If we now are issued with the knowledge that a is 90% likely to be greater than 1/2, how would we adjust our estimate?

3.1 Solution:

Here, we are given our qubit as:

$$|\psi\rangle = \cos\theta\,|0\rangle + \sin\theta\,|1\rangle$$

and our amplitude estimate of $\sin^2\theta$ is the amplitude squared of the state $|1\rangle$ which enables us to create our quantum circuit directly.

In quantum computation, this is a direct application to the amplitude estimation where we use the principles of quantum phase estimation and grover's algorithm to find the probability distribution of the amplitude estimator $a = \sin^2\theta$ with n copies of $|\psi\rangle$

(a)

Now, since we take n qubits, we can assume some unitary algorithm $\hat{A}$ and oracle $\hat{O}$ partitioning computational basis into 'good subset' G and 'bad' subset $B = \{|j\rangle_n\}_{j=1}^{2^n} - G$.

The state prepared by algorithm $\hat{A}$ to the reference state $|0\rangle^{\otimes n}$ without loss of generality is given by:

$$|A\rangle_n = \hat{A}\,|0\rangle^{\otimes n} = \sqrt{1-a}\,|A_B\rangle_n + \sqrt{a}\,|A_G\rangle_n$$

where the normalized states $|A_G\rangle$ is the $|1\rangle$ state and $|A_B\rangle$ is the $|0\rangle$ state.

The core of the quantum amplitude estimation algorithm uses a Grover-like iteration operator

$$\hat{Q} = \hat{A}(I - 2\,|0\rangle\langle 0|_n)\hat{A}^\dagger\hat{O}$$

(This is the culmination of the unitary tranformation $U = 2\,|\psi\rangle\langle\psi| - I$ and the unitary transformation $V = I - 2P$ for the projector P giving $a = \langle\psi|\,P\,|\psi\rangle$)

which can be shown to have a pair of eigenvalues $\lambda_\pm = e^{\pm 2i\theta}$ and $a = \sin^2\theta$.

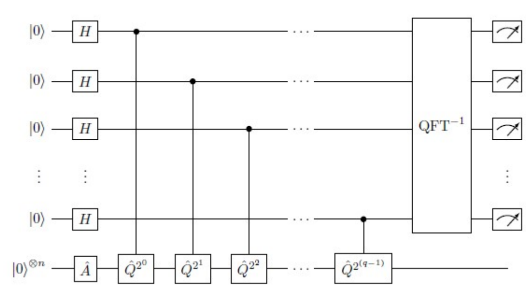

Figure 5: Circuit for quantum amplitude estimation

Building the circuit, we require the n-qubit register and an additional q-qubit control register and controlling applications of the Grover iteration $\hat{Q}$, which enable the tranformation of the results $m_0, m_1, \cdots, m_{q-1}$ into an integer $k = 2^0 m_0 + 2^1 m_1 + \cdots 2^{q-1} m_{q-1}$ (superposition) and converting it into an angle, result in either $\frac{\pi k}{2^q} = \theta_{\tilde{a}}$ or $\pi - \theta_{\tilde{a}}$ where $|\theta_{\tilde{a}} - \theta_a| = O(\frac{1}{2^q})$

thus, with high probability, the amplitude estimate

$$\tilde{a} = \sin^2(min[(\frac{\pi k}{2^q}, \pi - \frac{\pi k}{2^q})])$$

within an error $\epsilon \sim O(\frac{1}{2^q})$ of the true amplitude a.

We can see the estimate of a w.r.t algotihm output $\tilde{a}$ such that

$$|\tilde{a} - a| \leq 2\pi \frac{\sqrt{a(1-a)}}{q} + \frac{\pi^2}{q^2}$$

with the probability atleast $8/\pi^2$.

We can see the distribution of the amplitude estimate can be seen as a normal distribution with the specifics dependent on the number of iterations and the mean as the true amplitude a.

(b) Now, we are given that a is greater than 1/2 with 90% probability.

Thus, we can introduce Grover depth to our circuit which is defined as the parallel Grover computations on our qubit. Assuming d_j applications of the Grover iteration operator $\hat{Q}$ to the state $|A\rangle = \sin(\theta)|A_G\rangle + \cos(\theta)|A_B\rangle$ produces the state

$$|\psi_{d_j}\rangle = \hat{Q}^{d_j}|A\rangle = \sin[(2d_j + 1)\theta]|A_G\rangle + \cos[(2d_j + 1)\theta]|A_B\rangle$$

(where we know that $|A\rangle$ is our qubit $|\psi\rangle$ and $|A_G\rangle$ and $|A_B\rangle$ are $|1\rangle$ and $|0\rangle$)

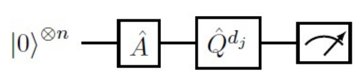

Figure 6: Circuit for maximum likelihood quantum amplitude estimation

Also, a computational-basis measurement then produces a 'good' state from G with probability

$$p_{d_j}(\theta) = \sin^2[(2d_j + 1)\theta]$$

and probability of the estimate is referred to as the likelihood is given by:

$$L(\theta = \theta_a; h_{d_j}) = [p_{d_j}(\theta_a)]^{h_{d_j}}[1 - p_{d_j}(\theta_a)]^{N_{shot} - h_{d_j}}$$

where h_{d_j} is the function of the possible values of θ.

Therefore we can limit for function for θ between $\pi/4$ and $\pi/2$ with a 90% likelihood and the determine the appropriate depth for a optimal estimate of a.

Graphically, it limits our peaks in the specified region for get the maximum-likelihood quantum amplitude estimate.

Also, the average additive error in the estimate $\tilde{a}$ will be approximately

$$\epsilon_{avg} = \sqrt{E[(\tilde{a} - a)^2]} \approx \sqrt{\frac{a(1-a)}{N_{shot} \sum_{j=0}^{q-1} (2d_j + 1)^2}}$$

4 4^{th} Question

A cohort of N graduate students need to be accomodated on a corridor having N equally spaced rooms. The students like to visit their friends' offices during the day. Each student has a set of friends amongst the others, each of whom they like to visit once a day; since students need "exercise" they will still go visit their friend even if the friend has already visited them. All students are equally lazy, so the cost of a visit is defined to be the distance between the two students' offices.

Write a function in Python that takes as input some convenient representation of the friendship relations between the students, and outputs an ordering of the students on the corridor such that the total cost of each day's visits as low as possible.

Notes:

- Write your code as you would production code
- Your solution need not be optimal. Cheap approximations are good.
- You should consider how the time and space requirements scale with N. Try to avoid exponential scaling.
- Please include explanations of the principles applied in your solution.

4.1 Solution:

We can look at the problem like optimizing the arrangement of the students to minimize the time of total friend visits according to their friendships.

We assume that the input is given to us in the form of a N x N matrix where the frienships values are 1 and thus the diagonal values are 0.

Although we can approach this problem by using an undirected graph or by using predefined functions, we take a direct approach by transforming the matrix to understand the computational process easily.

1. Assumption 1: Since we see that total time travelled by/for is directly proportional to the number of friends he has, we can initialize our list by placing the students with most number of friends in the middle of the list and the students with least friends at the edges of the list. (Initialization Function in code)

2. Assumption 2: We optimize our function by swapping order of people with same number of friends or with a friend more/less than them and compare the costs to find the optimum order for our function. (Optimum Function in code)

3. We build a cost function which calculates the total time cost of all the friends' visits done by the students. This is the function we try to minimize in our problem. (Total Function in code)

4. Since we can see that our matrix of friendships transforms according to the order of the students and thus we form a function to create the new friendship matrix according to which the cost can be calculated. (Transformation Function in code)

5. Moreover, we create an array which contains the friendship locations in the original friendship matrix which makes it easy to manipulate the answers. (frn array in main code)

Here, we also test the code with a sample friendship matrix a and find the optimum order for lowest cost.

```
import string
import numpy as np


def total_cost(a):  #this calculates the total time cost of a
    particular order of students
    n = len(a[0])
    cost = 0
    k = []
    for i in range(n):
        p=0
        for j in range(n):
            if (a[i][j] == 1):
                p+= abs(j-i)
        k.append(p)

    for i in k:
        cost+=i

    return cost


def init(a, lst):           #this function returns the list
    according to the no of friends a person has {with max friends
    people in the middle}

    s = []                  #this list contains the total number of
     friends a person has
    for i in range(n):
        p=0
        for j in range(n):
            if (a[i][j] == 1):
                p+= 1
        s.append(p)
```

```

    for i in range(n):       #here we sort the list in ascending order
        for j in range(i,n):
            if (s[j]< s[i]):
                temp1 = s[j]
                s[j] = s[i]
                s[i] = temp1
                temp2 = lst[j]
                lst[j] = lst[i]
                lst[i] = temp2


    i = 0
    j = 0
    k = n - 1
    lst_cpy = lst.copy()

    while (i < n):           # here we make the initial list based on our assumption

        lst_cpy[j] = lst[i]
        j+=1
        i+=1
        if (i< n):
            lst_cpy[k] = lst[i]
            k-=1
            i+=1

    return lst_cpy


def trans(pos, frn):        #this transformation function takes the input of the 1s from the main matrix and position changes and gives the new matrix

    #here we have the changed sequence of people and thus the changed 1s by refering to x as alphabets to nummbers
    x=[]
    for i in range(n):
        for j in range(n):
            if(pos[i] == stud[j]):
                x.append(j)

    mod = np.zeros((np.shape(frn)[0], 2))
    for i in range(np.shape(frn)[0]):
        for j in range(2):
            for k in range(n):
                if(frn[i][j]==x[k]):
                    mod[i][j]=k
                    break
    #now we create the new matrix

    new_mat = np.zeros((n,n))
    for k in range(np.shape(frn)[0]):
        new_mat[int(mod[k,0]), int(mod[k,1])] = 1
        new_mat[int(mod[k,1]), int(mod[k,0])] = 1
```

```

    return new_mat




def optim(mat, stu):

    s = []                  #this list contains the total number of friends a person has
    for i in range(n):
        p=0
        for j in range(n):
            if (mat[i][j] == 1):
                p+= 1
        s.append(p)



    temp_lst = stu.copy()
    new_lst = stu.copy()
    temp_mat = trans(stu, frn)
    new_mat = trans(stu, frn)

    for i in range(n):                        #this does the swapping and checks for the cost measurements and returns the optimum order
        for j in range(n):
            if(s[i]==s[j] or s[i]==s[j]+1 or s[i] == s[j]-1):
                temp_lst[i], temp_lst[j] = temp_lst[j], temp_lst[i]
                temp_mat = trans(temp_lst, frn)
                new_mat = trans(new_lst, frn)
                temp_mat_cost = total_cost(temp_mat)
                new_matr_cost = total_cost(new_mat)

                if (temp_mat_cost < new_matr_cost):
                    new_lst = temp_lst.copy()

    return new_lst







#main program

a=[[0,1,1,1,1],[1,0,1,1,1],[1,1,0,1,0],[1,1,1,0,1],[1,1,0,1,0]]  #this is the matrix that defines the friendships
n = len(a[0])
stud=[]               #this list contains the original order of students and named aphabetically
alpha = 'a'
for i in range(0, n):
    stud.append(alpha)
    alpha=chr(ord(alpha) + 1)
```

```python

new_stud = stud.copy()

ini = init(a, new_stud)                   #this contains the initializing order



l=0
frn = []                                          # here we have 1s i.e. the frienship locations
for i in range(n):
    for j in range(i,n):
        if (a[i][j] == 1):
            frn.append([i,j])
            l+=1

new_mat = trans(ini, frn)             #the matrix after the initializing function

order=ini.copy()

sol = optim(new_mat, order)       #here sol will contain our final optimum order for minimum cost

final_mat = trans(sol, frn)                              #this contains the optimum matrix of friendships
final_cost = total_cost(final_mat)                   #this is the minimum cost for the problem

print(sol, final_mat, final_cost)
```

Listing 1: List optimization code