Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 20, Number 2 (2024), pp. 253-260

http://www.ripublication.com

Quantum Algorithm for Clique Problem by Shor's Fourier Transform with RAM on QCEngine

Toru Fujimura

Art and Physical Education area security office, University of Tsukuba, Ibaraki-branch, Rising Sun Security Service Co., Ltd., 1-1-1, Tennodai, Tsukuba, Ibaraki 305-8577, Japan

Abstract

A quantum algorithm for the clique problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported. When k persons of n persons form the clique, these members are decided. A complexity of a classical computation is $n!/((n - k)!k!)$. The complexity of this Shor's Fourier transform method is able to be several times.

***Keywords*:** Quantum algorithm, clique problem, Shor's Fourier transform, RAM, QCEngine.

AMS subject classification: Primary 81-08; Secondary 68R10, 68W40.

1. INTRODUCTION

The clique problem has been discussed by Watanabe. [1] A quantum algorithm for the clique problem has been reported by Fujimura. [2] Still more, Fujimura discussed a quantum algorithm for the clique problem by the quantum Fourier transform on the QCEngine. [3]

According to my advanced study, when the clique problem is regarded as a special pattern of the knapsack problem, the complexity of the clique problem is able to be several times, because the knapsack problem is discussed by the Shor's Fourier transform. [4]

Therefore, the quantum algorithm for the clique problem is examined by the Shor's

Fourier transform with the RAM on the QCEngine, its result is reported.

2. CLIQUE PROBLEM

When k of n is forming the clique, these members are decided. A complexity of a classical computation is $n!/((n - k)!k!)$ times, because a number of the combinations chooses k from n, and its order is unrelated. [1-3, 5]

3. QUANTUM ALGORITHM

3-1. Knapsack Problem

As for n pieces of different weight luggage, the knapsack problem requests the best combination of the luggage packed into the knapsack that a weight r is assumed to be an upper bound. [5, 6]

When weights of the n pieces of luggage are assumed m_1, m_2, … , and m_n, and coefficients in which 0 or 1 are taken are x_1, x_2, … , and x_n, a sum of weights becomes $m_1x_1 + m_2x_2 + \ldots + m_nx_n$.

It can be said from the above-mentioned fact that the knapsack problem is a problem of requesting the best combination of 0 and 1 of x_1, x_2, … , and x_n in the upper bound weight r. [6]

3-2. From Knapsack Problem To Clique Problem

It is assumed that n is number of points, m is number of sides ($m \leq n(n - 1)/2$, and number of data qubits), and j is number of work qubits that included the sum of distances (distance is length between two points.).

First of all, query quantum registers $|x_i\rangle$ [$1 \leq i \leq m$. i and m are integers. m is a number of sides.], work1 quantum registers $|w_{1,j}\rangle$ [$1 \leq j \leq t$. j and t are integers. t is a necessary number for the sum of distances.], work2 quantum registers $|w_{2,p}\rangle$ [$1 \leq p \leq t$ +1. p and t are integers. t is a necessary number for the sum of distances. +1 is a qubit for the negative integer. [7]], and ancilla quantum qubits $|a_q\rangle$ [q is a necessary number for decrement with modulus.] are prepared.

Step 1: The distance data [$d(u_i, v_i)$: i-th distance between u_i and v_i. $u_i \neq v_i$. u_i and v_i are points.] are introduced to the RAM [7].

Step 2: Each qubit of $|x_i\rangle$, $|w_{1,j}\rangle$, $|w_{2,p}\rangle$, and $|a_q\rangle$ is set $|0\rangle$.

Step 3: The Hadamard gate H [1-11] acts on each qubit of $|x_i\rangle$. It changes them for entangled states.

Step 4: For $|x_i\rangle$, RAM [i - 1] [RAM has distance data of 0 → (m - 1).] is incremented

in $|w_{2,p}\rangle$. In a function, $F = \Sigma_{i=1 \to m} d(u_i, v_i)x_i$ is computed, where $d(u_i, v_i)$ is i-th distance. This operation makes entangled data base.

Step 5: For $|w_{2,p}\rangle$, mod(k) [k is a length of a route, and at random.] is done, where mod(k) is made by subtraction and addition in this program. [7] Therefore, the subtraction and the addition are done by necessary times, where work1 quantum registers are added work2 quantum registers, and the uncompute is done.

Step 6: For $|x_i\rangle$, the quantum Fourier transform (= QFT) [1, 3-5,7-9] is done.

Step 7: For $|x_i\rangle$ and $|w_{1,j}\rangle$, the proves are done.

Step 8: For $|x_i\rangle$, the read is done.

Step 9: A number of spikes is estimated by the function (https: //oreilly-qc. github. io? p = 12-4 [7]), where the function estimate_num_spikes (spike, range) [spike: read value, range: 2^m] is used.

Step 10: From candidates of the number of spikes, the repeat period P is obtained.

Step 11: From $P = 2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + \ldots + 2^{m-1}x_m$, when there is $k = d(u_1, v_1)x_1 + d(u_2, v_2)x_2 + \ldots + d(u_m, v_m)x_m$, it is answer [number of combination of necessary distances].

4. Example of Numerical Computation

It is assumed that 6 (= m) distances are $d(u_1, v_1) = d(1, 2) = 1$, $d(u_2, v_2) = d(1, 4) = 1$, $d(u_3, v_3) = d(2, 3) = 1$, $d(u_4, v_4) = d(2, 4) = 1$, $d(u_5, v_5) = d(3, 4) = 1$, $d(u_6, v_6) = d(4, 5) = 1$, and the upper bound of the length is 6, and $n = 5$. Furthermore, it is assumed that the mod(k) = mod(3), and query quantum register qubits $i = m = 6$. In this example, when mod(3) is 0, P is 0, 7, 11, 13, 14, 19, 25, 26, 28, 35, 37, 38, 41, 42, 44, 49, 50, 52, 56, and 63.

An example of program on the QCEngine is the following.

```
10 var a = [1, 1, 1, 1, 1, 1]; // RAM_a, distance data.
20 var query_qubits = 6;
30 var work1_qubits = 3;
40 var work2_qubits = 4;
50 var ancilla_qubits = 3; // subtractions of 3 times are able.
60 qc.reset(query_qubits + work1_qubits + work2_qubits + ancilla_qubits);
70 var query = qint.new(query_qubits, 'query');
80 var work1 = qint.new(work1_qubits, 'work1');
90 var work2 = qint.new(work2_qubits, 'work2');
100 var ancilla = qint.new(ancilla_qubits, 'ancilla');
```

```
110 qc.label('q'); // set query
120 query.write(0);
130 query.hadamard();
140 qc.label(' ');
150 qc.label('w1'); // set work1
160 work1.write(0);
170 qc.label('w2'); // set work2
180 work2.write(0);
190 qc.label('a'); // set ancilla
200 ancilla.write(0);
210 qc.print('RAM before increment: '+a+'¥n');
220 var query11 = 11; // one of query
230 var query28 = 28; // one of query
240 var k = 3; // upper bound of length
250 var work1_0 = 0; // one of work1. one of mod(k). k = 3.
260 qc.label('increment');
270 work2.add(a[0],query.bits(0x1));
280 work2.add(a[1],query.bits(0x2));
290 work2.add(a[2],query.bits(0x4));
300 work2.add(a[3],query.bits(0x8));
310 work2.add(a[4],query.bits(0x10));
320 work2.add(a[5],query.bits(0x20));
330 qc.label('mod(' + k + ')');
340 work2.subtract(k);
350 qc.cnot(ancilla.bits(0x1),work2.bits(0x8));
360 work2.add(k, ancilla.bits(0x1));
370 work2.subtract(k);
380 qc.cnot(ancilla.bits(0x2),work2.bits(0x8));
390 work2.add(k, ancilla.bits(0x2));
400 work2.subtract(k);
410 qc.cnot(ancilla.bits(0x4),work2.bits(0x8));
```

```
420 work2.add(k, ancilla.bits(0x4));
430 work1.add(work2);
440 qc.label('uncompute');
450 work2.subtract(k,ancilla.bits(0x4));
460 qc.cnot(ancilla.bits(0x4),work2.bits(0x8));
470 work2.add(k);
480 work2.subtract(k,ancilla.bits(0x2));
490 qc.cnot(ancilla.bits(0x2),work2.bits(0x8));
500 work2.add(k);
510 work2.subtract(k,ancilla.bits(0x1));
520 qc.cnot(ancilla.bits(0x1),work2.bits(0x8));
530 work2.add(k);
540 work2.subtract(a[5],query.bits(0x20));
550 work2.subtract(a[4],query.bits(0x10));
560 work2.subtract(a[3],query.bits(0x8));
570 work2.subtract(a[2],query.bits(0x4));
580 work2.subtract(a[1],query.bits(0x2));
590 work2.subtract(a[0],query.bits(0x1));
600 qc.label('QFT');
610 query.QFT();
620 var prob11 = 0; var prob28 = 0;
630 prob11 += query.peekProbability(query11); prob28 += query.peekProbability(query28);
640 // Print output query-Prob
650 qc.print(' Prob_query11: ' + prob11); qc.print(' Prob_query28: ' + prob28);
660 var prob0 = 0;
670 prob0 += work1.peekProbability(work1_0); // work1_0 = 0
680 // Print output work1-Prob
690 qc.print(' Prob_work1_0: ' + prob0);
700 // read
710 qc.label('Rq');
```

```
720 var b2 = query.read();
730 // Print output result
740 qc.print(' Read query = ' + b2 + '.');
750 // end
```

When this program is copied on Programming Quantum Computers https: //orelly-qc. github. io/# [free on-line quantum computation simulator QCEngine] [7], you can run it. [Caution!: Please delate the line numbers.]

A result of this program is the following.

The probe value of $|w_{1,j}\rangle = 0 : \approx 0.3437$.

The probe value of $|x_i\rangle = 11 : \approx 0.002464$.

The probe value of $|x_i\rangle = 28 : \approx 0.005203$.

The example of 10 times test: The read value of $|x_i\rangle$ = 55, 10, 42, 47, 55, 11, 23, 17, 43, 0. (= spike)

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range: $2^m = 2^6 = 64$]:

55 → 7, 14, 21, 28, 36, 43, 50 ; 10 → 6, 13, 19, 32 ; 42 → 3, 6, 9, 12, 15, 17, 20, 23, 26, 29, 32 ; 47 → 4, 8, 11, 15, 30, 34 ; 11 → 6, 12, 17, 23, 29, 35 ; 23 → 3, 6, 8, 11, 14, 25, 39 ; 17 → 4, 8, 11, 15, 30, 34 ; 43 → 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 34, 37, 40 ; 0 → nothing.

When P is 11 and 28, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6$:

$$2^0\times1 + 2^1\times1 + 2^2\times0 + 2^3\times1 + 2^4\times0 + 2^5\times0 = 11,$$

$$2^0\times0 + 2^1\times0 + 2^2\times1 + 2^3\times1 + 2^4\times1 + 2^5\times0 = 28.$$

There is $d(1, 2)x_1 + d(1, 4)x_2 + d(2, 3)x_3 + d(2, 4)x_4 + d(3, 4)x_5 + d(4, 5)x_6$:

$$d(1, 2)\times1 + d(1, 4)\times1 + d(2, 3)\times0 + d(2, 4)\times1 + d(3, 4)\times0 + d(4, 5)\times0$$

$$= d(1, 2) + d(1, 4) + d(2, 4) = 3 (= k),$$

$$d(1, 2)\times0 + d(1, 4)\times0 + d(2, 3)\times1 + d(2, 4)\times1 + d(3, 4)\times1 + d(4, 5)\times0$$

$$= d(2, 3) + d(2, 4) + d(3, 4) = 3 (= k).$$

5. DISCUSSION

In the knapsack problem, there are many combinations of luggages to obtain a value. In the clique problem, there are several combinations of persons to obtain a value. Therefore, the search is difficult.

In section 4, n points' graph has $m \leq n(n-1)/2$ bases. When N is 2^m, in the Grover's method, the complexity is $N^{1/2} = 2^{m/2}$, in the Shor's Fourier transform, it is several times.

In $m = 6$, $N^{1/2} = 2^3 = 8$, and several times $\approx 10/4 \approx 3$.

In this range, the Shor's Fourier transform is less than the complexity of the Grover's method.

6. SUMMARY

The quantum algorithm for the clique problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported.

The complexity of this method is several times.

I will apply this method for other problems.

REFERENCES

[1] Watanabe, Y., 2021, Nyumon Kogi Ryoshi Konpyuta (A Lecture of Introduction to Quantum Computer), Kodansha, Tokyo, Japan [in Japanese].

[2] Fujimura, T., 2012, "Quantum algorithm for clique problem," Glob. J. Pure Appl. Math., **8**, 175-182.

[3] Fujimura, T., 2024, "Quantum algorithm for clique problem by quantum Fourier transform on QCEngine," Glob. J. Pure Appl. Math., **20**, 61-69.

[4] Fujimura, T., 2023, "Quantum algorithm for knapsack problem by Shor's Fourier transform with RAM on QCEngine," Glob. J. Pure Appl. Math., **19**, 547-554.

[5] Takeuchi, S., 2005, Ryoshi Konpyuta (Quantum Computer), Kodansha, Tokyo, Japan [in Japanese].

[6] Fujimura, T., 2023, "Quantum algorithm for knapsack problem by usual Grover iteration with *z*-axis-rotation (= 180 degrees) on QCEngine," Glob. J. Pure Appl. Math., **19**, 23-29.

[7] Johnston, E. R., Harrigan, N., and Gimeno-Segovia, M., 2019, Programming Quantum Computers, O'Reilly, ISBN 978-1-492-03968-6.

[8] Shor, P. W., 1994, "Algorithm for quantum computation: discrete logarithms and factoring," Proc. 35th Annu. Symp. Foundations of Computer Science, IEEE, pp.124-134.

[9] Miyano, K., and Furusawa, A., 2008, Ryoshi Konpyuta Nyumon (An Introduction to Quantum Computation), Nipponhyoronsha, Tokyo, Japan [in Japanese].

[10] Grover, L. K., 1996, "A fast quantum mechanical algorithm for database search," Proc. 28th Annu. ACM Symp. Theory of Computing, pp.212-219.

[11] Grover, L. K., 1998, "A framework for fast quantum mechanical algorithms," Proc. 30th Annu. ACM Symp. Theory of Computing, pp.53-62.