Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 20, Number 2 (2024), pp. 323-330

http://www.ripublication.com

Quantum Algorithm for Hamilton Cycle 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 Hamilton cycle problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported. It is decided whether there is a route that all points is walked at once in a graph. When a number of points is n, the number of bases is $m \leq n(n - 1)/2$. In the quantum algorithm by the Shor's Fourier transform with the REM, its search is done by several times.

Keywords: Quantum algorithm, Hamilton cycle problem, Shor's Fourier transform, RAM, QCEngine.

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

1. INTRODUCTION

The Hamilton cycle problem has been discussed by Watanabe. [1] Fujimura discussed a quantum algorithm for the traveling salesman problem by the Shor's Fourier transform with the RAM on the QCEngine. [2]

According to my advanced study, when the Hamilton cycle problem is regarded as a special pattern of the traveling salesman problem, the complexity of the Hamilton cycle problem is able to be several times.

Therefore, the quantum algorithm for the Hamilton cycle problem is examined by the Shor's Fourier transform with the RAM on the QCEngine, its result is reported.

2. Hamilton Cycle Problem

It is decided whether there is a route that all points are walked at once in a graph. When a number of points is n, the number of bases is $m \leq n\,(n - 1)/2$. [1]

3. Quantum Algorithm

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 query 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. [3]], 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 [3].

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 [1-8] 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 $0 \rightarrow (m - 1)$.] is incremented in $|w_{2,p}\rangle$. In a function, $F = \Sigma_{i=1 \rightarrow m}\, d(u_i, v_i)x_i$ 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. [3] 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) [1-6] 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 [3]), 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 10 (= m) distances are $d(u_1, v_1) = d(1, 2) = 1$, $d(u_2, v_2) = d(2, 3) = 1$, $d(u_3, v_3) = d(3, 4) = 1$, $d(u_4, v_4) = d(4, 5) = 1$, $d(u_5, v_5) = d(5, 6) = 1$, $d(u_6, v_6) = d(6, 7) = 1$, $d(u_7, v_7) = d(7, 8) = 1$, $d(u_8, v_8) = d(8, 1) = 1$, $d(u_9, v_9) = d(1, 4) = 1$, $d(u_{10}, v_{10}) = d(5, 8) = 1$, and the upper bound of the length is 10, and $n = 8$. Furthermore, it is assumed that the mod(k) = mod(8), and query quantum register qubits $i = m = 10$. In this example, when mod(8) is 0, P is 0, 255, 383, 447, 479, 495, 503, 507, 509, 510, 639, 703, 735, 751, 759, 763, 765, 766, 831, 863, 879, 887, 891, 893, 894, 927, 943, 951, 955, 957, 958, 975, 983, 987, 989, 990, 1003, 1005, 1006, 1011, 1013, 1014, 1017, 1018, and 1020.

An example of program on the QCEngine is the following.

```
10 var a = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1]; // RAM_a, distance data.
20 var query_qubits = 10;
30 var work1_qubits = 4;
40 var work2_qubits = 5;
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 query255 = 255; // one of query
```

```
230 var k = 8; // one of length
240 var work1_0 = 0; // one of work1. one of mod(k). k = 8.
250 qc.label('increment');
260 work2.add(a[0],query.bits(0x1));
270 work2.add(a[1],query.bits(0x2));
280 work2.add(a[2],query.bits(0x4));
290 work2.add(a[3],query.bits(0x8));
300 work2.add(a[4],query.bits(0x10));
310 work2.add(a[5],query.bits(0x20));
320 work2.add(a[6],query.bits(0x40));
330 work2.add(a[7],query.bits(0x80));
340 work2.add(a[8],query.bits(0x100));
350 work2.add(a[9],query.bits(0x200));
360 qc.label('mod(' + k + ')');
370 work2.subtract(k);
380 qc.cnot(ancilla.bits(0x1),work2.bits(0x10));
390 work2.add(k, ancilla.bits(0x1));
400 work2.subtract(k);
410 qc.cnot(ancilla.bits(0x2),work2.bits(0x10));
420 work2.add(k, ancilla.bits(0x2));
430 work2.subtract(k);
440 qc.cnot(ancilla.bits(0x4),work2.bits(0x10));
450 work2.add(k, ancilla.bits(0x4));
460 work1.add(work2);
470 qc.label('uncompute');
480 work2.subtract(k,ancilla.bits(0x4));
490 qc.cnot(ancilla.bits(0x4),work2.bits(0x10));
500 work2.add(k);
510 work2.subtract(k,ancilla.bits(0x2));
520 qc.cnot(ancilla.bits(0x2),work2.bits(0x10));
530 work2.add(k);
```

```
540 work2.subtract(k,ancilla.bits(0x1));
550 qc.cnot(ancilla.bits(0x1),work2.bits(0x10));
560 work2.add(k);
570 work2.subtract(a[9],query.bits(0x200));
580 work2.subtract(a[8],query.bits(0x100));
590 work2.subtract(a[7],query.bits(0x80));
600 work2.subtract(a[6],query.bits(0x40));
610 work2.subtract(a[5],query.bits(0x20));
620 work2.subtract(a[4],query.bits(0x10));
630 work2.subtract(a[3],query.bits(0x8));
640 work2.subtract(a[2],query.bits(0x4));
650 work2.subtract(a[1],query.bits(0x2));
660 work2.subtract(a[0],query.bits(0x1));
670 qc.label('QFT');
680 query.QFT();
690 var prob255 = 0;
700 prob255 += query.peekProbability(query255);
710 // Print output query-Prob
720 qc.print(' Prob_query255: ' + prob255);
730 var prob0 = 0;
740 prob0 += work1.peekProbability(work1_0); // work1_0 = 0
750 // Print output work1-Prob
760 qc.print(' Prob_work1_0: ' + prob0);
770 // read
780 qc.label('Rq');
790 var b2 = query.read();
800 // Print output result
810 qc.print(' Read query = ' + b2 + '.');
820 // end
```

When this program is copied on Programming Quantum Computers https: //orelly-qc. github. io/# [free on-line quantum computation simulator QCEngine] [3], you can

run it. [Caution!: Please delate the line numbers.]

A result of this program is the following.

The probe value of $|w_{1,j}\rangle = 0 : \approx 0.04492$.

The probe value of $|x_i\rangle = 255 : \approx 0.0002489$.

The example of 10 times test: The read value of $|x_i\rangle$ = 513, 0, 132, 4, 731, 1008, 0, 510, 265, 36. (= spike)

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range: $2^m = 2^{10} = 1024$]:

513 → even numbers from 2 to 244, 254, and odd numbers from 257 to 511 ; 0 → nothingness ; 132 → 8, 16, 23, 31, 62, 93, 124, 132, 163, 194, 225, 256, 512, 768 ; 4 → 256, 512, 768 ; 731 → 4, 7, 14, 21, 28, 35, 42, 49, 56, 63, 70, 77, 84, 91, 98, 105, 112, 119, 126, 133, 140, 147, 154, 161, 168, 171, 178, 185, 192, 199, 206, 213, 220, 227, 234, 241, 248, 255, 262, 269, 276, 283, 290, 297, 304, 311, 318, 325, 332, 339, 678, 685 ; 1008 → 64, 128, 192, 256, 320, 384, 448, 512, 576, 640, 704, 768, 832, 896, 960 ; 510 → even numbers from 2 to 126, odd numbers from 129 to 253, 255, 257, 512 ; 265 → 4, 8, 12, 15, 19, 23, 27, 54, 58, 85, 170, 228, 313, 626, 711 ; 36 → 28, 57, 114, 142, 199, 256, 512, 768.

When P is 255, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6 + 2^6x_7 + 2^7x_8 + 2^8x_9 + 2^9x_{10}$:

$2^0\times1 + 2^1\times1 + 2^2\times1 + 2^3\times1 + 2^4\times1 + 2^5\times1 + 2^6\times1 + 2^7\times1 + 2^8\times0 + 2^9\times0 = 255$.

There is $d(1, 2)x_1 + d(2, 3)x_2 + d(3, 4)x_3 + d(4, 5)x_4 + d(5, 6)x_5 + d(6, 7)x_6 + d(7, 8)x_7 + d(8, 1)x_8 + d(1, 4)x_9 + d(5, 8)x_{10}$:

$d(1, 2)\times1 + d(2, 3)\times1 + d(3, 4)\times1 + d(4, 5)\times1 + d(5, 6)\times1 + d(6, 7)\times1 + d(7, 8)\times1 + d(8, 1)\times1 + d(1, 4)\times0 + d(5, 8)\times0 = d(1, 2) + d(2, 3) + d(3, 4) + d(4, 5) + d(5, 6) + d(6, 7) + d(7, 8) + d(8, 1) = 8$ (= k).

5. Discussion

In the traveling salesman problem, when the shortest value is obtained, there is only one combination of distances (distance is length between two points).

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 $N^{1/2}$, in the Shor's Fourier transform, it is several times.

In $n = 8$ and $m = 10 \leq n(n-1)/2 = 8\times7/2 = 28$, $N^{1/2} = 2^{m/2} = 2^5 = 32$, and (several times) = 10/2 = 5.

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 Hamilton cycle 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., 2024, "Quantum algorithm for traveling salesman problem by Shor's Fourier transform with RAM on QCEngine," Glob. J. Pure Appl. Math., **20**, 143-150.

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

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

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

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

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

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