Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 19, Number 2 (2023), pp. 547-554

http://www.ripublication.com

Quantum Algorithm for Knapsack 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
E-mail: tfujimura8@gmail.com

Abstract

A quantum algorithm for the knapsack problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported. In this method, there is no using QRAM (= Quantum RAM), but only RAM is used. Its search is done by several times and high speed.

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

AMS subject classification: Primary 81-08; Secondary 81-10, 68Q12.

1 Introduction

Takeuchi discussed the complexity of the knapsack problem. [1] The quantum algorithm for the knapsack problem by the usual Grover iteration with z-axis-rotation (180 degrees) on the QCEngine (Quantum Computer Simulator [2]) was reported by Fujimura. [3] In this time, I used the Shor's Fourier transform [1, 2, 4, 5] with the RAM on the QCEngine for this problem. This method is done for its search by several times and high speed.

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

2 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 k is assumed to be an upper bound. [1, 3]

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 k. [3]

3 Quantum Algorithm

It is assumed that n is number of data qubits (= number of luggage), and j is number of weight qubits that included the sum of weights.

First of all, query quantum registers (= query registers) $|x_i\rangle$ [$1 \leq i \leq n$. i and n are integers. n is a number of luggage.], weight1 quantum registers (= weight1 registers) $|w_{1,j}\rangle$ [$1 \leq j \leq t$. j and t are integers. t is a necessary number for the sum of weight.], weight2 quantum registers (= weight2 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 weight. +1 is a qubit for the negative integer. [2]], and ancilla quantum qubits (= ancilla qubits) $|a_q\rangle$ [q is a necessary number for decrement with modulus.] are prepared.

Step 1: The weight data [$m_i : 1 \leq i \leq n$. i and n are integers. n is a number of luggage.] are introduced to the RAM [2].

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 [1-7] 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 weight data of 0 → (n - 1). They are $m_1 \rightarrow m_n$.] is incremented in $|w_{2,p}\rangle$. In a function, $F = \Sigma_{i=1 \rightarrow n} m_i x_i$ is computed, where m_i is weight. This operation makes entangled data base.

Step 5: For $|w_{2,p}\rangle$, mod(k) [k is the upper bound weight.] is done, where mod(k) is made by subtraction and addition in this program. [2] Therefore, the subtraction and the addition are done by necessary times, where weight1 registers are changed by CNOT weight2 registers, and the uncompute is done.

Step 6: For $|x_i\rangle$, the quantum Fourier transform (= QFT) [1, 2, 4, 5] 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 [2]), where the function estimate_num_spikes (spike, range) [spike: read value, range: 2^n] is used.

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

Step 11: From $P = 2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + \ldots + 2^{n-1}x_n$, when there is $k = m_1x_1 + m_2x_2 + \ldots + m_nx_n$, it is answer [number of combination of necessary luggage].

4. Example of Numerical Computation

It is assumed that 6 (= n) pieces of luggage of weight are m_1 = 13kg, m_2 = 8kg, m_3 = 3kg, m_4 = 6kg, m_5 = 15kg, m_6 = 2kg, and the upper bound of the weight of the knapsack is k = 23kg. Furthermore, it is assumed that the mod(k) = mod(23), t = 6 ($2^6 - 1 = 63$. Because, total sum is $\Sigma_{i=1 \to n} m_i = 47$.), and query register qubits n = 6. In this example, when mod(23) is 0, P is 0, 18, 35, and 56.

An example of program on the QCEngine is the following.

```
10 var a = [13,8,3,6,15,2]; // RAM_a, weight data.
20 var query_qubits = 6;
30 var weight1_qubits = 6;
40 var weight2_qubits = 7;
50 var ancilla_qubits = 3; // subtractions of 3 times are able.
60 qc.reset(query_qubits + weight1_qubits + weight2_qubits + ancilla_qubits);
70 var query = qint.new(query_qubits, 'query');
80 var weight1 = qint.new(weight1_qubits, 'weight1');
90 var weight2 = qint.new(weight2_qubits, 'weight2');
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 weight1
160 weight1.write(0);
170 qc.label('w2'); // set weight2
180 weight2.write(0);
190 qc.label('a'); // set ancilla
200 ancilla.write(0);
210 qc.print('RAM before increment: '+a+'¥n');
220 var query18 = 18; // one of query
230 var k = 23; // upper bound of knapsack
240 var weight1_0 = 0; // one of weight1. one of mod(k). k = 23.
250 qc.label('increment');
```

```
260 weight2.add(a[0],query.bits(0x1));
270 weight2.add(a[1],query.bits(0x2));
280 weight2.add(a[2],query.bits(0x4));
290 weight2.add(a[3],query.bits(0x8));
300 weight2.add(a[4],query.bits(0x10));
310 weight2.add(a[5],query.bits(0x20));
320 qc.label('mod(' + k + ')');
330 weight2.subtract(k);
340 qc.cnot(ancilla.bits(0x1),weight2.bits(0x40));
350 weight2.add(k, ancilla.bits(0x1));
360 weight2.subtract(k);
370 qc.cnot(ancilla.bits(0x2),weight2.bits(0x40));
380 weight2.add(k, ancilla.bits(0x2));
390 weight2.subtract(k);
400 qc.cnot(ancilla.bits(0x4),weight2.bits(0x40));
410 weight2.add(k, ancilla.bits(0x4));
420 qc.cnot(weight1.bits(0x1),weight2.bits(0x1));
430 qc.cnot(weight1.bits(0x2),weight2.bits(0x2));
440 qc.cnot(weight1.bits(0x4),weight2.bits(0x4));
450 qc.cnot(weight1.bits(0x8),weight2.bits(0x8));
460 qc.cnot(weight1.bits(0x10),weight2.bits(0x10));
470 qc.cnot(weight1.bits(0x20),weight2.bits(0x20));
480 qc.label('uncompute');
490 weight2.subtract(k,ancilla.bits(0x4));
500 qc.cnot(ancilla.bits(0x4),weight2.bits(0x40));
510 weight2.add(k);
520 weight2.subtract(k,ancilla.bits(0x2));
530 qc.cnot(ancilla.bits(0x2),weight2.bits(0x40));
540 weight2.add(k);
550 weight2.subtract(k,ancilla.bits(0x1));
560 qc.cnot(ancilla.bits(0x1),weight2.bits(0x40));
```

```
570 weight2.add(k);
580 weight2.subtract(a[5],query.bits(0x20));
590 weight2.subtract(a[4],query.bits(0x10));
600 weight2.subtract(a[3],query.bits(0x8));
610 weight2.subtract(a[2],query.bits(0x4));
620 weight2.subtract(a[1],query.bits(0x2));
630 weight2.subtract(a[0],query.bits(0x1));
640 qc.label('QFT');
650 query.QFT();
660 var prob18 = 0;
670 prob18 += query.peekProbability(query18);
680 // Print output query-Prob
690 qc.print(' Prob_query18: ' + prob18);
700 var prob0 = 0;
710 prob0 += weight1.peekProbability(weight1_0); // weight1_0 = 0
720 // Print output weight1-Prob
730 qc.print(' Prob_weight1_0: ' + prob0);
740 // read
750 qc.label('Rq');
760 var b2 = query.read();
770 // Print output result
780 qc.print(' Read query = ' + b2 + '.');
790 // end
```

When this program is copied on Programming Quantum Computers https: //orelly-qc. github. io/# [free on-line quantum computation simulator QCEngine] [2], you can run it. [Caution!: Please delate the line numbers.]

A result of this program is the following.

The probe value of $|w_{1,j}\rangle = 0 : \approx 0.06250$.

The probe value of $|x_i\rangle = 18 : \approx 0.01931$.

The example of 10 times test: The read value of $|x_i\rangle$ = 11, 32, 47, 33, 15, 54, 25, 7, 61, 11. (= spike)

The candidates of number of spikes are estimated by the function [the function

estimate_num_spikes (spike, range) [spike: read value, range: $2^n = 2^6 = 64$]:

11 → 6, 12, 17, 23, 29, 35 ; 32 → 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28, 30 ; 47 → 4, 8, 11, 15, 30, 34 ; 33 → 2, 4, 6, 8, 10, 12, 14, 17, 19, 21, 23, 25, 27, 29, 31 ; 15 → 4, 9, 13, 17, 34, 47 ; 54 → 6, 13, 19, 32 ; 25 → 3, 5, 10, 13, 18, 23 ; 7 → 9, 18, 27, 37, 46, 55 ; 61 → 21, 43.

When P is 18 and 35, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6$:

$2^0\times0 + 2^1\times1 + 2^2\times0 + 2^3\times0 + 2^4\times1 + 2^5\times0 = 18$,

$2^0\times1 + 2^1\times1 + 2^2\times0 + 2^3\times0 + 2^4\times0 + 2^5\times1 = 35$.

There is $m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + m_5x_5 + m_6x_6$:

$13\times0 + 8\times1 + 3\times0 + 6\times0 + 15\times1 + 2\times0 = 13\times1 + 8\times1 + 3\times0 + 6\times0 + 15\times0 + 2\times1 = 23$ (= k).

For $n = 4$:

RAM; a = [13, 8, 3, 6], $2^4 = 16$, mod(k) = mod(16), where query_qubits = 4, weight1_qubits = 4, weight2_qubits = 5, query5 = 5, and necessary changes are done. In this example, when mod(5) is 0, P is 0 and 5.

The probe value of $|w_{1,j}\rangle = 0 : \approx 0.1250$.

The probe value of $|x_i\rangle = 5 : \approx 0.01919$.

The example of 10 times test: The read value of $|x_i\rangle$ = 10, 3, 4, 0, 0, 7, 14, 1, 8, 6.

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range $2^n = 2^4 = 16$]:

10 → 3, 5, 8 ; 3 → 5, 11 ; 4 → 4, 8 ; 0 → nothingness ; 7 → 2, 5, 7 ; 14 → 8 ; 1 → nothingness ; 8 → 2, 4, 6 ; 6 → 3, 5, 8.

When P is 5, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4$:

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

There is $m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4$:

$13\times1 + 8\times0 + 3\times1 + 6\times0 = 16$ (= k).

5 Discussion

Section 4 is general example. In this section, special example is examined.

For $n = 6$, RAM ; a = [1, 2, 4, 8, 16, 32], $2^6 = 64$, mod(k) = mod(16), above-mentioned program is done. In this example, when mod(16) is 0, P is 0, 16, 32, and 48.

Its result is the following.

The probe value of $|w_{1,j}\rangle = 0$: 0.06250.

The probe value of $|x_i\rangle = 16$: 0.06250.

The example of 10 times test: The read value of $|x_i\rangle$ = 44, 4, 12, 8, 20, 28, 56, 44, 0, 44. (= spike)

The candidates of number of spikes are estimated by the function:

44 → 3, 6, 10, 13, 16, 32 ; 4 → 16, 32, 48 ; 12 → 5, 11, 16, 32, 48 ; 8 → 8, 16, 24, 32, 40, 48 ; 20 → 3, 6, 10, 13, 16, 32 ; 28 → 2, 5, 7, 9, 16, 32 ; 56 → 8, 16, 24, 32, 40, 48 ; 0 → nothingness. When P is 16, 32, and 48, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6$:

$2^0\times0 + 2^1\times0 + 2^2\times0 + 2^3\times0 + 2^4\times1 + 2^5\times0 = 16$,

$2^0\times0 + 2^1\times0 + 2^2\times0 + 2^3\times0 + 2^4\times0 + 2^5\times1 = 32$, and

$2^0\times0 + 2^1\times0 + 2^2\times0 + 2^3\times0 + 2^4\times1 + 2^5\times1 = 48$.

There is $m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + m_5x_5 + m_6x_6$:

$1\times0 + 2\times0 + 4\times0 + 8\times0 + 16\times1 + 32\times0 = 16$ (= k),

$1\times0 + 2\times0 + 4\times0 + 8\times0 + 16\times0 + 32\times1 = 32$ (= $2k$), and

$1\times0 + 2\times0 + 4\times0 + 8\times0 + 16\times1 + 32\times1 = 48$ (= $3k$).

All of answers are obtained.

6 Summary

The quantum algorithm for the knapsack 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. It is less than the complexity of Grover's method. Therefore, its search is done by high speed.

I will apply this method for other problems.

References

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

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

[3] 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.

[4] Shor, P. W., 1994, "Algorithm for quantum computation: discrete logarithms and

factoring," Proc. 35th Annu. Symp. Foundations of Computer Science, IEEE, pp.124-134.

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

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

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