Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 20, Number 2 (2024), pp. 233-239
1© Research India Publications
http://www.ripublication.com

Quantum Algorithm for Bin-packing 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 bin-packing problem by the Shor's Fourier transform with the RAM on the QCEngine, and its example are reported. It is decided whether n pieces of luggage are packed into u boxes, where each weight of the luggage is k or less, and a maximum storage weight of the box is k. A complexity of a classical calculation is u^n. The complexity becomes several times by the quantum algorithm that uses the Shor's Fourier transform.

Keywords: Quantum algorithm, bin-packing problem, Shor's Fourier transform, RAM, QCEngine.

AMS subject classification: Primary 81-08; Secondary 68R05, 68W40.

1. INTRODUCTION

A quantum algorithm for the bin-packing problem by the Grover iteration has been discussed by Fujimura. [1] Still more, Fujimura discussed a quantum algorithm for the knapsack problem by the Shor's Fourier transform with the RAM on the QCEngine. [2]

According to my advanced study, when the bin-packing problem is regarded as a special pattern of the knapsack problem, the complexity of the bin-packing problem is able to be several times.

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

2. BIN-PACKING PROBLEM

It is decided whether n pieces of luggage are packed into u boxes, where each weight of the luggage is k or less, and a maximum storage weight of box is k. [1, 3] A complexity of a classical calculation is u^n because there are u choices about each luggage.

It can be said that the bin-packing 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. [4]

3. QUANTUM ALGORITHM

It is assumed that n pieces of luggage and u boxes, where each weight of the luggage is k or less, and the maximum storage weight of the box is k. When each weight of the luggage is m_1, m_2, … , m_n, and coefficients in which 0 or 1 are taken are x_1, x_2, … , and x_n, a sum of weights becomes $m_1x_1 + m_2x_2 + \ldots + m_nx_n$. Still more, j is number of weight qubits that included the sum of weights.

First of all, query quantum registers $|x_i\rangle$ [$1 \leq i \leq n$. i and n are integers. n is a number of luggage.], weight1 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 weight.], weight2 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 weight. +1 is a qubit for the negative integer. [5]], and ancilla quantum 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 [5].

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, 2, 4-10] 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 $0 \rightarrow (n - 1)$. 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. [5] Therefore, the subtraction and the addition are done by necessary times, where weight1 quantum registers are added weight2 quantum registers, and the uncompute is done.

Step 6: For $|x_i\rangle$, the quantum Fourier transform (= QFT) [2, 5-8] 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 [5]), 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 7 (= n) pieces of luggage of weight are m_1 = 7kg, m_2 = 6kg, m_3 = 5kg, m_4 = 4kg, m_5 = 3kg, m_6 = 3kg, m_7 = 2kg, and the upper bound of the weight of the box is k = 10kg, and u is 3. Furthermore, it is assumed that the mod (k) = mod (10), t = 5 ($2^5 - 1 = 31$. Because, total sum is $\Sigma_{i=1 \rightarrow n} m_i = 30$.), and query quantum register qubits n = 7. In this example, when mod (10) is 0, P is 0, 10, 17, 27, 33, 43, 56, 71, 84, 94, 100, 110, 117, and 127.

An example of program on the QCEngine is the following.

```
10 var a = [7, 6, 5, 4, 3, 3, 2]; // RAM_a, weight data.
20 var query_qubits = 7;
30 var weight1_qubits = 5;
40 var weight2_qubits = 6;
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 query10 = 10; // one of query
230 var k = 10; // upper bound of knapsack
240 var weight1_0 = 0; // one of weight1. One of mod (k). k = 10.
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 weight2.add (a [6], query. Bits (0x40));
330 qc.label ('mod (' + k + ')');
340 weight2.subtract (k);
350 qc.cnot (ancilla.bits (0x1), weight2.bits (0x20));
360 weight2.add (k, ancilla.bits (0x1));
370 weight2.subtract (k);
380 qc.cnot (ancilla.bits (0x2), weight2.bits (0x20));
390 weight2.add (k, ancilla.bits (0x2));
400 weight2.subtract (k);
410 qc.cnot (ancilla.bits (0x4), weight2.bits (0x20));
420 weight2.add (k, ancilla.bits (0x4));
430 weight1.add (weight2);
440 qc.label ('uncompute');
450 weight2.subtract (k,ancilla.bits (0x4));
460 qc.cnot (ancilla.bits (0x4), weight2.bits (0x20));
470 weight2.add (k);
480 weight2.subtract (k,ancilla.bits (0x2));
490 qc.cnot (ancilla.bits(0x2),weight2.bits(0x20));
500 weight2.add (k);
```

```
510 weight2.subtract (k,ancilla.bits(0x1));
520 qc.cnot (ancilla.bits (0x1), weight2.bits (0x20));
530 weight2.add (k);
540 weight2.subtract (a [6], query. Bits (0x40));
550 weight2.subtract (a [5], query. Bits (0x20));
560 weight2.subtract (a [4], query. Bits (0x10));
570 weight2.subtract (a [3], query. Bits (0x8));
580 weight2.subtract (a [2], query. Bits (0x4));
590 weight2.subtract (a [1], query. Bits (0x2));
600 weight2.subtract (a [0], query. Bits (0x1));
610 qc.label ('QFT');
620 query.QFT ();
630 var prob10 = 0;
640 prob10 += query.peekProbability (query10);
650 // Print output query-Prob
660 qc.print (' Prob_query10: ' + prob10);
670 var prob0 = 0;
680 prob0 += weight1.peekProbability (weight1_0); // weight1_0 = 0
690 // Print output weight1-Prob
700 qc.print (' Prob_weight1_0: ' + prob0);
710 // read
720 qc.label ('Rq');
730 var b2 = query.read ();
740 // Print output result
750 qc.print (' Read query = ' + b2 + '.');
760 // end
```

When this program is copied on Programming Quantum Computers https: //orelly-qc. Github. Io/# [free on-line quantum computation simulator QCEngine] [5], you can run it. [Caution! Please delate the line numbers.]

A result of this program is the following.

The probe value of $|w_{1,j}\rangle = 0$: ≈ 0.1094.

The probe value of $|x_i\rangle = 10$: ≈ 0.002345.

The example of 10 times test: The read value of $|x_i\rangle$ = 69, 106, 59, 88, 57, 78, 39, 55, 27, 123. (= spike)

The candidates of number of spikes are estimated by the function [the function estimate_num_spikes (spike, range) [spike: read value, range: $2^n = 2^7 = 128$]:

69 → 2, 4, 7, 9, 11, 13, 26, 39, 52, 65 ; 106 → 6, 12, 17, 23, 29, 35, 64 ; 59 → 2, 4, 7, 9, 11, 13, 26, 39, 52, 65 ; 88 → 3, 6, 10, 13, 16, 32, 48, 64, 80 ; 57 → 2, 5, 7, 9, 18, 27, 36, 45, 54, 63, 65 ; 78 → 3, 5, 10, 13, 18, 23, 41, 64 ; 39 → 3, 7, 10, 13, 23, 40, 69, 82 ; 55 → 2, 5, 7, 14, 21, 28, 35, 42, 49, 56, 63, 65 ; 27 → 5, 10, 14, 19, 38, 57, 71, 90 ; 123 → 26, 51, 77.

When P is 10, 17, 27, 56, and 71, $2^0x_1 + 2^1x_2 + 2^2x_3 + 2^3x_4 + 2^4x_5 + 2^5x_6 + 2^6x_7$:

$2^0\times0 + 2^1\times1 + 2^2\times0 + 2^3\times1 + 2^4\times0 + 2^5\times0 + 2^6\times0 = 10$,

$2^0\times1 + 2^1\times0 + 2^2\times0 + 2^3\times0 + 2^4\times1 + 2^5\times0 + 2^6\times0 = 17$,

$2^0\times1 + 2^1\times1 + 2^2\times0 + 2^3\times1 + 2^4\times1 + 2^5\times0 + 2^6\times0 = 27$,

$2^0\times0 + 2^1\times0 + 2^2\times0 + 2^3\times1 + 2^4\times1 + 2^5\times1 + 2^6\times0 = 56$,

$2^0\times1 + 2^1\times1 + 2^2\times1 + 2^3\times0 + 2^4\times0 + 2^5\times0 + 2^6\times1 = 71$.

There is $m_1x_1 + m_2x_2 + m_3x_3 + m_4x_4 + m_5x_5 + m_6x_6 + m_7x_7$:

$7\times0 + 6\times1 + 5\times0 + 4\times1 + 3\times0 + 3\times0 + 2\times0 = 10$ (= k),

$7\times1 + 6\times0 + 5\times0 + 4\times0 + 3\times1 + 3\times0 + 2\times0 = 10$ (= k),

$7\times1 + 6\times1 + 5\times0 + 4\times1 + 3\times1 + 3\times0 + 2\times0 = 20$ (= $2k$),

$7\times0 + 6\times0 + 5\times0 + 4\times1 + 3\times1 + 3\times1 + 2\times0 = 10$ (= k),

$7\times1 + 6\times1 + 5\times1 + 4\times0 + 3\times0 + 3\times0 + 2\times1 = 20$ (= $2k$).

Therefore, (6, 4), (7, 3), and (5, 3, 2) are obtained.

5. DISCUSSION

In the knapsack problem, there are many combinations of luggage to obtain a value. In the bin-packing problem, when there are many boxes, the search is difficult.

In section 4, the number of boxes are 3 (= u). In the Grover's method, the complexity is about $(u^n = 3^7)^{1/2} \approx 47$. In the Shor's Fourier transform, the complexity is several times.

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 bin-packing 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] Fujimura, T., 2011, "Quantum algorithm for bin-packing problem," Glob. J. Pure Appl. Math., **7**, 383-386.

[2] 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.

[3] Weisstein, E. W., 2010, "Bin-packing problem," [Online], Available: http: //mathworld. Wolfram. Com /Bin-PackingProblem. Html.

[4] 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.

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

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

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

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

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

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