Global Journal of Pure and Applied Mathematics.
ISSN 0973-1768 Volume 19, Number 1 (2023), pp. 151-156

http://www.ripublication.com/gjpam.htm

Quantum Algorithm for 3-SAT Problem by Grover Iteration with CCCNOT Gate (=Control Toffoli Gate) 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 3-SAT problem by the Grover iteration with the CCCNOT gate (=control Toffoli gate) on the QCEngine, and its example are reported. In this method, there are 3 literals with 2 'OR's in each clause that is used the CCCNOT gate. The times of iterations are about $(\pi/4)(2^n/m)^{1/2}$, where n is a number of qubits, and m is a number of marked terms. This method is simple and powerful.

Keywords: Quantum algorithm, 3-SAT problem, Grover iteration, CCCNOT gate (=control Toffoli gate), QCEngine.

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

1. Introduction

The 3-SAT problem has been discussed by Cook for the complexity. [1] Quantum computer's example of 3-SAT problem is reported by Johnston, Harrigan, and Gimeno-Segovia with QCEngine (free on-line quantum computer simulator). [2]
According to my advanced study, when there are 3 literals with 2 "OR"s in each clause, one clause is able to program by one "OR" gate operator. This method is simple and powerful.
Therefore, because the quantum algorithm for the 3-SAT problem is examined by the Grover iteration with CCCNOT gate (=control Toffoli gate) on the QCEngine, its result is reported.

2. 3-SAT Problem

In the 3-SAT problem, it is assumed that (i) each value of n variables becomes "TRUE" or "FALSE", "~" is "NOT", "V" is "OR", "&" is "AND", (ii) "V", "~", and 3 different variables are included in each parentheses (=clause) that are connected by "&". If a value of logical formula by the literals and the logical connectives is "TRUE", it is decided whether there is at least one combination of values of the variables or not. [1, 2]

3. Quantum Algorithm

The following conditions are assumed. (I) Each value of variables x_1, x_2, …, x_{n-1}, and x_n becomes "TRUE" [=1] or "FALSE" [=0]. "~" is "NOT". "V" is "OR". "&" is "AND". For example, it is assumed in this algorithm that (1 V 1 V 1), (1 V 1 V 0), and (1 V 0 V 0) become 1, and (0 V 0 V 0) becomes 0. (II) "V", "~", and 3 different variables in x_1, x_2, …, x_{n-1}, and x_n are included in each clause, and then the clauses are connected by "&". In these conditions, if a value of logical formula by the literals and the operators is "TRUE", it is searched whether there is at least one combination of values of the variables or not. It is assumed that n is number of qubits.

First of all, query quantum registers $|x_i\rangle$ [$3 \leq i \leq n$. i is an integer.] and work quantum registers (=ancilla qubits) $|w_j\rangle$ [$1 \leq j \leq u$. j and u are integers. u is a necessary number for work.] are prepared.

Step 1: Each qubit of $|x_i\rangle$ and $|w_j\rangle$ is set $|0\rangle$.

Step 2: The Hadamard gate H [2-7] acts on each qubit of $|x_i\rangle$. It changes them for entangled states.

Step 3: Each clause is presented by $|x_i\rangle$, $|w_j\rangle$, CCCNOT gate, and quantum operators.

Step 4: For $|w_j\rangle$, the flip is done. The one-marked-term's rotation angle is 180 degrees by z-axis.

Step 5: Uncomputation is done.

Step 6: For $|x_i\rangle$, Grover-iteration is done.

Step 7: For $|w_j\rangle$ and $|x_i\rangle$, the probes are done.

Step 8: Step 3 → 7 are returned by about $(\pi/4)(2^n/m)^{1/2}$ times [2] [m is a number of marked terms.].

Step 9: Each of $|x_i\rangle$ is read. The one-marked-term is obtained.

4. Example of Numerical Computation

For example at n =4, it is assumed that the one-marked-term =8, logical formula: (x_1 V x_2 V x_4) & (~x_1 V x_2 V x_3) & (x_1 V ~x_2 V x_4) & (~x_2 V x_3 V x_4) & (x_1 V ~x_3 V x_4) & (~x_1 V x_2 V ~x_3) & (~x_2 V x_3 V ~x_4) & (x_1 V ~x_3 V ~x_4) & (~x_1 V ~x_2 V ~x_3) & (~x_1 V ~x_2 V ~x_4), and each value of $x_{1\sim4}$: x_1 =x_2 =x_3 =0, x_4 =1, and work qubits =u =10.

An example of program on the QCEngine is the following.

```
10 var query_qubits =4;
20 var work_qubits =10;
30 qc.reset(query_qubits + work_qubits);
40 var query =qint.new(query_qubits, 'query');
```

```
50 var work =qint.new(work_qubits, 'work');
60 qc.label('s q');
70 query.write(0);
80 query.hadamard();
90 qc.label(' ');
100 qc.label('s w');
110 work.write(0);
120 var query8 =8;
130 var work0 =0;
140 var number_of_iterations =4;
150 for (var i =0; i < number_of_iterations; ++i)
160 {
170 qc.label('Gate');
180 qc.not(query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
190 qc.cnot(work.bits(0x1), query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
200 qc.not(query.bits(0x1)|query.bits(0x2)|query.bits(0x8)|work.bits(0x1));
210 qc.not(query.bits(0x2)|query.bits(0x4));
220 qc.cnot(work.bits(0x2), query.bits(0x1)|query.bits(0x2)|query.bits(0x4));
230 qc.not(query.bits(0x2)|query.bits(0x4)|work.bits(0x2));
240 qc.not(query.bits(0x1)|query.bits(0x8));
250 qc.cnot(work.bits(0x4), query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
260 qc.not(query.bits(0x1)|query.bits(0x8)|work.bits(0x4));
270 qc.not(query.bits(0x4)|query.bits(0x8));
280 qc.cnot(work.bits(0x8), query.bits(0x2)|query.bits(0x4)|query.bits(0x8));
290 qc.not(query.bits(0x4)|query.bits(0x8)|work.bits(0x8));
300 qc.not(query.bits(0x1)|query.bits(0x8));
310 qc.cnot(work.bits(0x10), query.bits(0x1)|query.bits(0x4)|query.bits(0x8));
320 qc.not(query.bits(0x1)|query.bits(0x8)|work.bits(0x10));
330 qc.not(query.bits(0x2));
340 qc.cnot(work.bits(0x20), query.bits(0x1)|query.bits(0x2)|query.bits(0x4));
350 qc.not(query.bits(0x2)|work.bits(0x20));
360 qc.not(query.bits(0x4));
370 qc.cnot(work.bits(0x40), query.bits(0x2)|query.bits(0x4)|query.bits(0x8));
380 qc.not(query.bits(0x4)|work.bits(0x40));
390 qc.not(query.bits(0x1));
400 qc.cnot(work.bits(0x80), query.bits(0x1)|query.bits(0x4)|query.bits(0x8));
410 qc.not(query.bits(0x1)|work.bits(0x80));
420 qc.cnot(work.bits(0x100), query.bits(0x1)|query.bits(0x2)|query.bits(0x4));
430 qc.not(work.bits(0x100));
440 qc.cnot(work.bits(0x200), query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
450 qc.not(work.bits(0x200));
460 qc.label('Flip');
470 work.cphase(180, 0x1|0x2|0x4|0x8|0x10|0x20|0x40|0x80|0x100|0x200);
480 qc.label('Uncompute');
490 qc.not(work.bits(0x200));
```

```
500 qc.cnot(work.bits(0x200), query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
510 qc.not(work.bits(0x100));
520 qc.cnot(work.bits(0x100), query.bits(0x1)|query.bits(0x2)|query.bits(0x4));
530 qc.not(query.bits(0x1)|work.bits(0x80));
540 qc.cnot(work.bits(0x80), query.bits(0x1)|query.bits(0x4)|query.bits(0x8));
550 qc.not(query.bits(0x1));
560 qc.not(query.bits(0x4)|work.bits(0x40));
570 qc.cnot(work.bits(0x40), query.bits(0x2)|query.bits(0x4)|query.bits(0x8));
580 qc.not(query.bits(0x4));
590 qc.not(query.bits(0x2)|work.bits(0x20));
600 qc.cnot(work.bits(0x20), query.bits(0x1)|query.bits(0x2)|query.bits(0x4));
610 qc.not(query.bits(0x2));
620 qc.not(query.bits(0x1)|query.bits(0x8)|work.bits(0x10));
630 qc.cnot(work.bits(0x10), query.bits(0x1)|query.bits(0x4)|query.bits(0x8));
640 qc.not(query.bits(0x1)|query.bits(0x8));
650 qc.not(query.bits(0x4)|query.bits(0x8)|work.bits(0x8));
660 qc.cnot(work.bits(0x8), query.bits(0x2)|query.bits(0x4)|query.bits(0x8));
670 qc.not(query.bits(0x4)|query.bits(0x8));
680 qc.not(query.bits(0x1)|query.bits(0x8)|work.bits(0x4));
690 qc.cnot(work.bits(0x4), query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
700 qc.not(query.bits(0x1)|query.bits(0x8));
710 qc.not(query.bits(0x2)|query.bits(0x4)|work.bits(0x2));
720 qc.cnot(work.bits(0x2), query.bits(0x1)|query.bits(0x2)|query.bits(0x4));
730 qc.not(query.bits(0x2)|query.bits(0x4));
740 qc.not(query.bits(0x1)|query.bits(0x2)|query.bits(0x8)|work.bits(0x1));
750 qc.cnot(work.bits(0x1), query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
760 qc.not(query.bits(0x1)|query.bits(0x2)|query.bits(0x8));
770 qc.label('Grover');
780 query.hadamard();
790 query.not();
800 query.cphase(180);
810 query.not();
820 query.hadamard();
830 var prob0 =0;
840 prob0 +=work.peekProbability(work0);
850 // Print output work-Probs
860 qc.print(' Prob_work0: ' + prob0);
870 var prob8 =0;
880 prob8 +=query.peekProbability(query8);
890 // Print output query-Probs
900 qc.print(' Prob_query8: ' + prob8);
910 }
920 //read
930 qc.label('Rq');
940 var b2 =query.read();
```

```
950 // Print output result
960 qc.print(' Read query =' + b2 +'.');
970 //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, where T is the time of Grover iterations.
The probe value of $|w_j\rangle$ =0: 1.0000 [T =1 → 4].
The probe value of $|x_i\rangle$ =8: T =1; 0.4727, T =2; 0.9084, T =3; 0.9613, T =4; 0.5817.
The read of $|x_i\rangle$ =8.
Therefore, the best times of Grover iterations are 3 for n =4.

5. Discussion

In this time, when each clause includes 3 literals with 2 "OR"s, (for example) the program is qc. not (query. bits (0x1)| query. bits (0x2)| query. bits (0x4)); → qc. cnot (work. bits (0x1), query. bits (0x1)| query. bits (0x2)| query. bits (0x4)); → qc. not (query. bits (0x1)| query. bits (0x2)| query. bits (0x4)| work. bits (0x1));.
If each clause includes k literals with (k-1) "OR"s [k > 3. k is an integer.], (for example) the program is qc. not(query. bits (0x1)| query. bits (0x2)| … | query. bits (0x[address number of qubit for k-th literal])); → qc. cnot (work. bits (0x1), query. bits (0x1)| query. bits (0x2)| … | query. bits (0x[address number of qubit for k-th literal])); → qc. not (query. bits (0x1)| query. bits (0x2)| … | query. bits (0x[address number of qubit for k-th literal])| work. bits (0x1));.

6. Summary

When each clause includes 3 literals with 2 "OR"s, simple program that includes CCCNOT gate (=control Toffoli gate) is done, and the times of Grover iterations are about $(\pi/4)(2^n/m)^{1/2}$ [n is a number of query qubits, and m is a number of marked terms.].
This method is simple and powerful.
I will apply this method for other problems.

References

[1] Cook, S. A., 1971, "The complexity of theorem proving procedures, " Proc. 3rd Annu. ACM Symp. Theory of Computing, pp.151-158
[2] Johnston, E.R., Harrigan, N., and Gimeno-Segovia, M., 2019, Programing Quantum Computers, O'Reilly, ISBN 978-1-492-03968-6.
[3] Takeuchi, S., 2005, Ryoshi Konpyuta (Quantum Computer), Kodansha, Tokyo, Japan [in Japanese].
[4] Grover, L. K., 1996, "A fast quantum mechanical algorithm for database search, " Proc. 28th Annu. ACM Symp. Theory of Computing, pp.212-219.

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

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