An Implementation of Compact Genetic Algorithm on a Quantum Computer

Sorrachai Yingchareonthawornchai[1] , Chatchawit Aporntewan[2] , Prabhas Chongstitvatana[1]
[1]Department of Computer Engineering
[2]Department of Mathematics and Computer Science
Chulalongkorn University
Phayathai Rd., Pathumwan, Bangkok, Thailand, 10330
sorrachai.y@student.chula.ac.th, {chatchawit.a, prabhas.c}@chula.ac.th

Abstract— **Programming a quantum computer posts a challenge. It is not straight forward to transfer the current programming skill on a classical computer to a quantum computer. This work presents an example of programming a quantum computer. The compact genetic algorithm is used as a target as it is powerful and popular method in evolutionary computation. A quantum bit (qubit) concept was introduced as a basis for storing information. The representation of quantum register has benefits over classical computing, i.e. the quantum operation allows manipulating qubits in the way that it is impossible in a classical computer. This paper demonstrates the enhancement in terms of solution quality and speed introduced by quantum computation. The simulation of quantum computing is carried out for solving a problem using the compact genetic algorithm.**

Keywords: Genetic algorithms, Compact Genetic Algorithms, Quantum Computers.

I. INTRODUCTION

Quantum computers achieve speedup over classical computers by taking the advantages of the interference between quantum amplitudes. This phenomenon is hard to simulate in classical computers. There have been many examples of quantum computation that outperform classical computing such as Grover's search algorithm [1] and Shor's fast factoring algorithm [2]. On the other hand, a genetic algorithm (GA) is basically a search algorithm. It belongs to a class of evolutionary computation. The key idea is based on the principle of biological evolution, such as natural selection, genetic inheritance and mutation. There have been the relevant attempts between quantum and genetic algorithms. For example, Quantum Genetic Optimization Algorithm [3] has introduced an optimization of classical genetic algorithm using the principles of quantum search which provided a significant speed-up on each genetic step. There are many variations of genetic algorithms. The compact genetic algorithm (cGA) [4] is one of them. In a classical computer, the compact genetic algorithm represents the population as a probability distribution over the set of solutions by using a vector. In a quantum computer, the population is represented as a probability distribution in a quantum register. This paper demonstrates an alternative way to program a quantum computer to perform compact genetic algorithms. The study uses QCL (Quantum Computation Language) [5]—[7] as an emulator of quantum computer.

II. COMPACT GENETIC ALGORITHMS

Genetic algorithms are adaptive search algorithms based on the idea of biological evolution such as natural selection, cross over and mutation. Compact genetic algorithm represents the population using a vector. The vector contains each bit with a real number from 0.0 to 1.0 representing the probability of that bit to be one. This reduces the storage of the population to just the storage of a vector. This property makes it very suitable to be implemented in a quantum computer by representing a vector using a quantum register.

Here is a short description of the steps in cGA. The first step is to generate a population. An appropriate encoding of the candidate solution is dependent on the problem. The second step is to sample two candidates from the population and evaluates their fitness using the fitness function in order to provide the fitness value of each candidate. The third step is to determine the "winner" by comparing their fitness values. The winner's chromosome will be used to update the probability vector so that the distribution will converge to a population that fits the solution requirement. This is an iterative process. The process will continue until the terminating condition is met.

III. INTRODUCTION TO QUANTUM COMPUTATION.

A. Definition of a quantum bit

A quantum bit or a qubit is a unit of information describing a two-dimensional quantum system. A qubit is represented as 2-by-1 matrix with a complex number, as

$$|0\rangle = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \tag{1}$$

$$|1\rangle = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \tag{2}$$

The two basis state can be superposed,

$$|\varphi\rangle = \begin{pmatrix} \alpha \\ \beta \end{pmatrix} = \alpha\begin{pmatrix} 1 \\ 0 \end{pmatrix} + \beta\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \alpha|0\rangle + \beta|1\rangle \tag{3}$$

with the condition that,

$$|\alpha|^2 + |\beta|^2 = 1 \quad (4)$$

The $|\alpha|^2$ is the probability that the measurement of state will result in state $|0\rangle$, and the $|\beta|^2$ is the probability that the measurement of state will result in state $|1\rangle$. Keep in mind that the general qubit cannot be seen: whenever the qubit is measured or observed, it spontaneously become a bit. Next, a representation of a qubit is introduced

B. The Bloch Sphere.

The general state of one qubit system can be represented in the form,

$$|\varphi\rangle = \alpha|0\rangle + \beta|1\rangle \quad (5)$$

Where α and β are complex numbers. It might seem there are four parameters. However, the equation holds the condition that,

$$|\alpha|^2 + |\beta|^2 = 1 \quad (6)$$

So, the equation can be reformed in terms of two parameters,

$$|\varphi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\phi}\sin\frac{\theta}{2}|1\rangle \quad (7)$$

With natural ranges $0 \leqslant \theta \leqslant \pi$ and $0 \leqslant \phi \leqslant 2\pi$.

As only two real numbers are required to represent a qubit, it can be mapped into a three-dimensional coordinate system. The mapping looks like a unit sphere known as Bloch Sphere. See Fig.1.

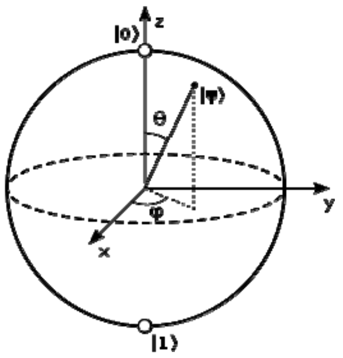

Figure 1. Bloch Sphere.

C. Quantum Operation

1) Measurement: The measure command measures the quantum register and returns the measured value. The measure operation is not reversible.

2) Unitary Gates:

a) Hadamard Gate: The Hadamard Gate is defined by the transformation matrix,

$$H = \frac{1}{\sqrt{2}}\begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix} \quad (8)$$

The usage is to map n qubits initialized with $|0\rangle$ to a superposition of all 2^n orthogonal states in the $|0\rangle, |1\rangle$ basis with equal weight. This means, if observed, the state will collapse to be a 0 or 1 with equal probability.

b) Qubit Rotation: The rotation of a single qubit is defined by the transformation matrix,

$$U(\theta) = \begin{pmatrix} \cos\frac{\theta}{2} & \sin\frac{\theta}{2} \\ -\sin\frac{\theta}{2} & \cos\frac{\theta}{2} \end{pmatrix} \quad (9)$$

This operator is implemented as
`Rot(real theta,qureg q);`

The `theta` parameter used in this paper has range from $\frac{\pi}{2}$ to $-\frac{\pi}{2}$

c) Using Hadamard gate and Qubit Rotation: Hadamard gate is used for transforming into an even superposition state of quantum register. According to Bloch's sphere, a qubit can be visualized as Fig.2-left. When a qubit rotation is applied, for example pi/4, the state will be Fig.2-right. This means the qubit is more likely to be 0, when measure, than 1.

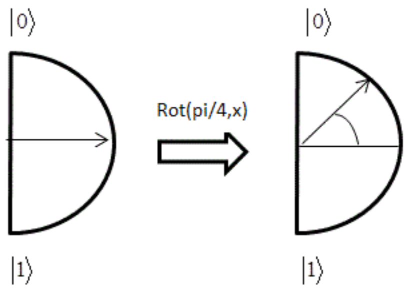

Figure 2. Applying Rot(pi/4,x)

IV. The Mapping of Compact Genetic Algorithm – Quantum Construction

The main contribution of this paper is the quantum construction of the compact genetic algorithm. The algorithm is described below:

Step 1) Initialize qureg x with Hadamard transform and record the vector v.

Step 2) Generate two individuals from qureg by measurement

```
resetregister(x);
H(x);
restore(x,v);
a := measure(x);
resetregister(x);
```

```
        H(x);
        restore(x,v);
        b := measure(x);
  Step 3) Let them compete
        {win, lose} := eval(a,b);
  Step 4) Update qureg x with the "winner"
       for i := 1 to n do
        if win[i] != lose[i] then
   if win[i] = 1  and v[i] != -1*NHALFPI then v[i] := v[i] –1;
   if win[i] = 0  and v[i] != 1*NHALFPI then v[i] := v[i] +1;
  Step 5) check loop time
       If time < LOOP_TIME then
                   time := time+1;
                   go to Step 2)
       else go to Step  6)
  Step 6) generate the final result
        resetregister(x);
        H(x);
        restore(x,v);
        output := measure(x);

  operator restore(qureg x, vector v)
  for i := 0 to n-1 do
    Rot( THETA * v[i], x[i]);

  parameter:
  const pi = 3.14159265
  const n = length
  const THETA = small angle ,e.g. pi/32
```

The first step is to initialize a quantum register as a quantum variable x. Then, apply Hadamard gate in order to produce an even superposition state. Record vector v as a vector of integer for tracking the operation. It will be used in *restore* operator.

In step 2, the measurement operation is used to generate an individual. The individual will be randomly produced according to probability distribution in a quantum variable. As a result, the quantum variable will be collapsed to a single state. However, the second individual must be produced. This can be done by resetting quantum register into the initial state, i.e. zero; applying Hadamard-gate, and calling restore procedure. The restore procedure iterates every bit to apply Rotation operation by using v[i]*THETA as a parameter. The THETA is a constant of small angle. The constant NHALFPI means an integer that v[i]*THETA equals to pi/2. The NHALFPI is used for bounding the condition to prevent the rotation "overflow."

In step 3, the eval function is called to compare the fitness values between individual a and individual b. The higher fitness individual will be denoted as a "winner."

In step 4, this is where the tracking operation is used to store integer. We iterate all bits. Notice that if a bit of winner is set, the value is decremented and vice versa. The reason is the output of the Rotation operation will go to 0 when the parameter is pi/2. Conversely, the output of the Rotation operation will go to 1 if the parameter is –pi/2.

V. Experimental Result For The Mapping

One-max problem is used to validate the quantum cGA. The one-max problem is a simple benchmark in GA. Its purpose is to search for the binary string with all ones. Only the fitness function, counting ones, is used to guide the search. Assume an individual A competes with individual B.

Individual	chromosome	fitness
A	1011	3
B	1001	2

The winner is A. The quantum variable is then updated toward the winner via qubit rotation. An experiment is set up with the following parameters: n = 5; LOOP_TIME = 150; THETA = pi/32. This is a 5-qubit quantum register. The execution is iterated 150 steps with the increment angle pi/32.

The Fig.3 was captured at i = 0, 10, 20, 30, 50 and 150 respectively.

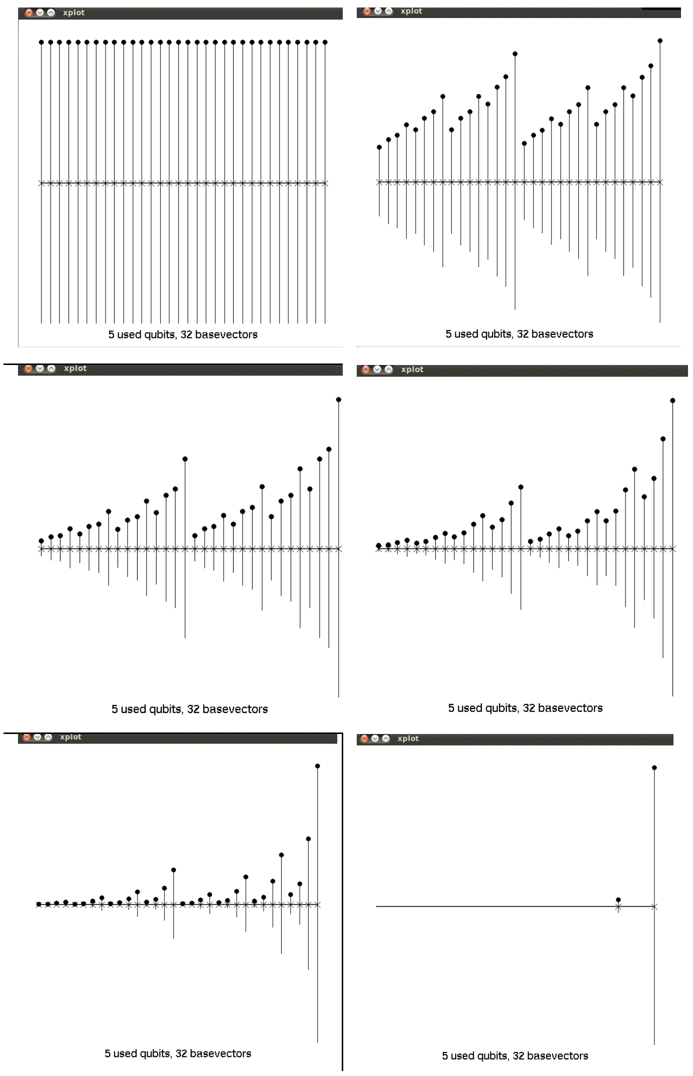

Figure 3. Plotting quantum variable

Fig.3 shows the probability distribution of a 5-qubit register. The x axis represents the basis state starting from 0 to 31 toward the right hand side. The y axis represents the probability of each state. Notice that at i = 0, it is starting from even superposition state of quantum variable. For each iteration, the Rotation operation will rotate the probability up to the winner of each stage. The direction will be toward to the one-max value, i.e. 31 which is 11111_2.

VI. The Compact Genetic Algorithm – Quantum Algorithm Construction

In order to achieve benefits over a classical computer, a quantum parallelism must be realized. The mapping of compact genetic algorithm into quantum computers has demonstrated that is at least computationally equivalent to classical computers. This section introduces enhanced version of compact genetic algorithm using quantum computation. The algorithm is described below:

```
Step 1) Initialize qureg x with Hadamard transform, and
a single qubit qureg y and record the vector v.
Step 2) Generate the first individual from qureg by
measurement
        resetregister(x);
        H(x);
        restore(x,v);
        a := measure(x);
Step 3) Generate the second individual with the
condition that the fitness is greater than or equal to the first
individiual
        resetregister(x&y);
        H(x);
        restore(x,v);
        Fit(x,y,a);
        c := measure(y);
      if c = 0 then go to Step 3)
      if c = 1 then
          b := measure(x);
          go to Step 4)
Step 4) Let them compete
        {win, lose} := eval(a,b);
Step 5) Update qureg x with the "winner"
       for i := 1 to n do
        if win[i] != lose[i] then
  if win[i] = 1 and v[i] != -1*NHALFPI then v[i] := v[i] –1;
  if win[i] = 0 and v[i] != 1*NHALFPI then v[i] := v[i] +1;
Step 6) check loop time
       If time < LOOP_TIME then
                 time := time+1;
                 go to Step 2)
       else go to Step 7)
Step 7) generate the final result
        resetregister(x);
        H(x);
        restore(x,v);
        output := measure(x);
```

```
operator restore(qureg x, vector v)
for i := 0 to n-1 do
  Rot( THETA * v[i], x[i]);

qufunct Fit(qureg x,qureg y,int a)
for each state s in x do
   if (fitness of s >= fitness of a) then Not(y);

parameter:
const pi = 3.14159265
const n = length
const THETA = small angle ,e.g. pi/32
```

The main concept of the algorithm is to generate the second individual based on the first individual. The fitness of the second individual is greater than or equal to the fitness of first individual. The tendency of second individual will be based on the greater than or equal one. Regardless of direction of the gradient, the evolution continues toward the solution.

The first step is the same as previous algorithm including a 1-qubit register y is initialized.

In step 3, this is perhaps the most interesting part. The quantum function [5] is introduced. A quantum function is a pseudo-classic operator with the characteristic:

$$F : |x\rangle_x |0\rangle_y \rightarrow |x\rangle_x |f(x)\rangle_y \qquad (10)$$

In this step, *Fit* quantum function is merely:

$$Fit : |x\rangle_x |0\rangle_y \rightarrow \begin{matrix} |x\rangle_x |1\rangle & if \quad fit(x) \geq fit(a) \\ |x\rangle_x |0\rangle & otherwise \end{matrix} \qquad (11)$$

By using this function, all population is divided into two groups: the greater or equal and the lower. The greater group is obtained by measuring the 1-qubit quantum register y. If the measure value is not one, repeat the process until the value one is obtained. Notice that a quantum function used here is performed on every possible state at once: achieving quantum parallelism.

VII. Experimental Result For Quantum Algorithm Construction

Trap functions are deceptive. It is hard to find the global optimum of these functions by using GAs. In this experiment, the two-peak trap function is used as a benchmark of effectiveness of the algorithm. An experiment is set up with the following parameters: n = 6; LOOP_TIME = 150; THETA = pi/64. The trap function is as Fig.4

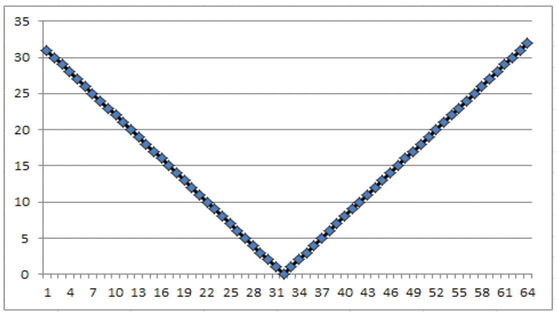

Figure 4. Two-peak trap.There is a false maximum at i = 0 with fitness of 31

The Fig.5 was captured at i = 5, 10, 20, 40, 80 and 120.

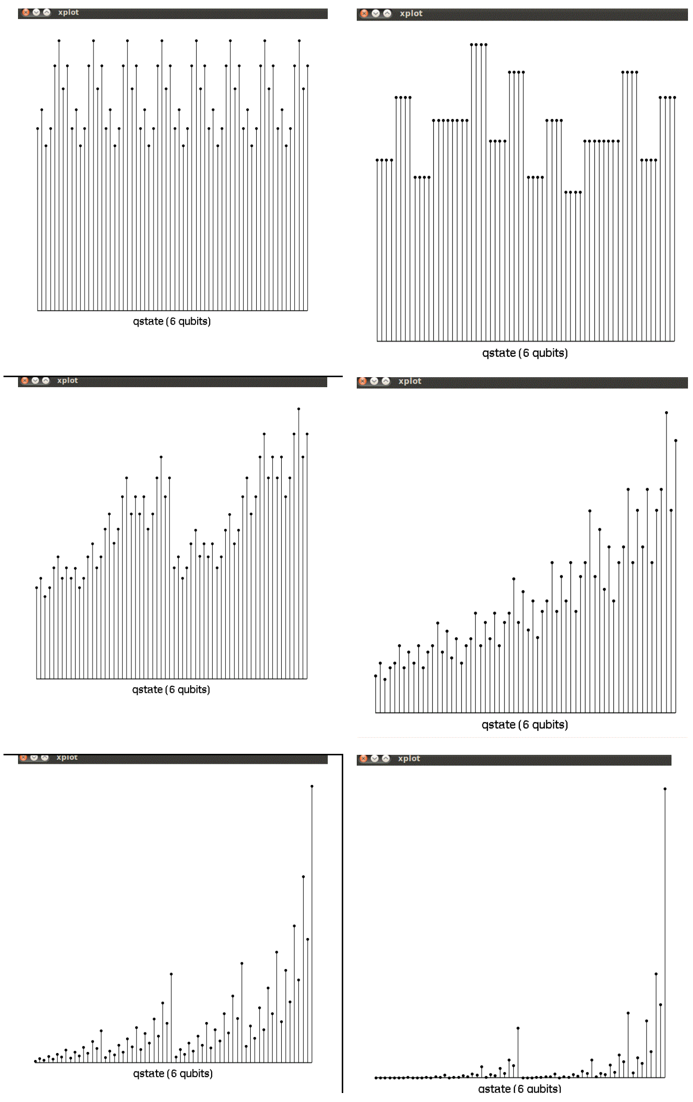

Figure 5. Plotting 6-qubit quantum variable

The Fig.5 shows the probability distribution of a 6-qubit register. The x axis represents the basis state starting from 0 to 31 toward the right hand side. The y axis represents the probability of each state. Notice that there are two side of tendency which is false peak and real maximum. The tendency of second individual will be based on the greater than or equal the first one. Regardless of direction of the gradient, the evolution continues toward the solution. Here is the solution quality of this algorithm compared to the mapping version. The fitness function used in the table was two-peak trap as the Fig.4. The accuracy was calculated from the percentage of corrected outputs running 100 times. Notice that the proposed algorithm has slightly benefited over the mapping version. However, the solution quality can be improved by increasing the number of selection iteration of quantum function.

TABLE I. COMPARISON OF THE SOLUTION QUALITY

qubits	**the Mapping of cGA**			**Proposed Algorithm**		
	1	***2***	***3***	***1***	***2***	***3***
4	87%	86%	87%	91%	91%	90%
5	68%	71%	69%	76%	78%	80%
6	57%	54%	52%	68%	64%	64%

The proposed quantum algorithm exploits the power of quantum parallelism. Also, the algorithm shows that the second individual is guided by the first individual. It will speed up the coverage toward the solution. Therefore, the experiment has achieved an enhancement in terms of speed and quality.

VIII. CONCLUSION

There are two quantum program presented here. The first one is the mapping of compact genetic algorithm into quantum computers which is similar to the version running on a classical computer. The whole population is stored in a quantum register. The running time complexity of quantum cGA is the same as a classical one. In order to achieve benefits over a classical computer, a quantum parallelism must be realized. The work presented here has achieved this goal. The later quantum program is the enhancement version using quantum computation

What this work has demonstrated is a kind of quantum computation which introduces the enhancement in terms of solution quality and speed.

REFERENCES

[1] Grover L.K.: A fast quantum mechanical algorithm for database search, Proceedings, 28th Annual ACM Symposium on the Theory of Computing, (May 1996) p. 212.

[2] P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer, SIAM J. Computing 26, pp. 1484-1509 (1997).

[3] Andrea Malossini, Enrico Blanzieri, T. Calarco: Quantum Genetic Optimization. IEEE Trans. Evolutionary Computation 12(2): 231-241 (2008).

[4] Georges R. Harik, Fernando G. Lobo, David E. Goldberg: The compact genetic algorithm. IEEE Trans. Evolutionary Computation 3(4): 287-297 (1999).

[5] Bernhard Ömer. A procedural formalism for quantum computing. Master's thesis, Department of Theoretical Physics, Technical University of Vienna, 1998.

[6] Bernhard Ömer. Quantum programming in QCL. Master's thesis, Institute of Information Systems, Technical University of Vienna, 2000.

[7] Bernhard Ömer. Structured Quantum Programming. PhD thesis, Technical University of Vienna, 2003.

[8] Nielsen, Michael A. and Chuang, Isaac L. Quantum Computation and Quantum Information: 10th Anniversary Edition. Cambridge : Cambridge University Press, 2010. 978-1-107-00217-3.

[9] Yanofsky, Noson S. and Mannucci, Mirco A. Quantum Computing for Computer Scientists. Cambridge : Cambridge University Press, 2008. 978-0-521-879965.

[10] R.P. Feynman, R.B. Leighton, and M. Sands. The Feynman Lectures on Physics, volume III. Addison-Wesley, 1965b.