Quantum-Evolutionary Algorithms: A SW-HW approach

D. PORTO, A. MARTINEZ, S. SCIMONE, E. SCIAGURA
AST Automotive
STMicroelectronics S.r.l.
Contrada Blocco Torrazze, C.P. 421, 95121 Catania
ITALY

Abstract: - In this paper a HW/SW platform for the implementation of an optimizer based on a evolutionary algorithm, called quantum-inspired evolutionary algorithm (QEA), is introduced. It is based on the concept and principles of quantum computing, such as *quantum bit* and *superposition of states*, whose features are briefly described. The hardware implementation of the QEA using a FPGA board is therefore described together with the use of a customable software fitness in order to solve general purpose problems. HW-SW connections are provided by a PCI interface. Final aim is to build a flexible object able to optimize the choices of some sensible parameters in a dynamic system quickly, with particular attention on industrial and automotive applications.

Key-Words: - QEA, Evolutionary Algorithms, Q-bit, Quantum computing, FPGA, Hardware Design.

1 Introduction

In the past decade, the solutions of some complex optimization problems have been dealt by Genetic Algorithms, a very effective and versatile optimization strategy. They work by "evolving" a set of potential solutions according to survival rules that give advantages to the best individuals. Due to this structure, it can be avoided the fall in local maxima or minima [1][2]. The accuracy of these methodologies is often paid with a high computational time, especially in solving complex problems. The recent coming of Quantum-Inspired Evolutionary Algorithm (QEA [3][4]) seems to overcome this limitation, denoting the same reliability and robustness of Genetic Algorithms but faster performances. Moreover, a hardware implementation of QEA could provide a further acceleration, giving us a tool able to perform online optimization of parameters in any dynamic system, such as combustion engines or industrial plants.
The object of this article is the realization of a hardware to perform the several operations composing a QEA and to maintain, however, a certain flexibility to handle problems as various as possible. At last the algorithm will be described from a theoretical point of view and it will be shown in the implemented details on the board.

2 QEA Overview

As Genetic Algorithms, QEA exploits the concepts of individual, population, evaluation of fitness, evolutions of the population, hence, the idea of generation. Since Quantum Computing adopts the notion of *Q-bit* and the principle of *superposition of states*, QEA, instead of the classic binary representation, use a Q-bit, defined as the smallest unit of information, in which there is a coexistence of the states 1 and 0, each one with its probability. The sum of the squares of these probabilities is 1. We assume a single individual (*Q-individual*) as a sequence of Q-bits. In this context, a Q-individual represents a linear superposition of the states 1 and 0, in the probabilistic search space. A new concept is the *Q-gate*, a variation operator which drives the evolution towards the best solution and towards a unique state. In fact, at the beginning, the QEA contains a population of one Q-individual that represents the linear superposition of all possible states with the same probability. As the probability of each Q-bit approaches either 1 or 0 by the Q-gate, the Q-individual converges to a single state and the diversity property disappears gradually. Compared to classical Genetic Algorithms, it has been seen that the QEA have better performances, for that concern processing time.
The results show that QEA performs very well even with small populations, without premature convergence as compared to the conventional Genetic Algorithms. Finally, we tell that QEA is not a Quantum Algorithm, but a novel Evolutionary Algorithm for a classical computer [5], [6].

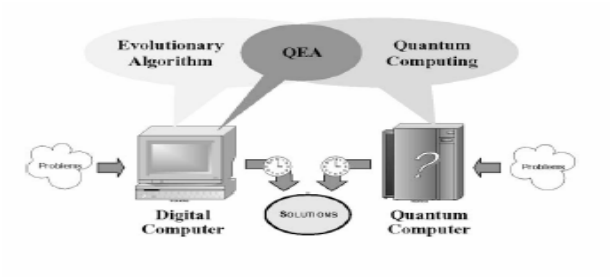

Figure 1: Simplified structure of QEA

3 Encoding and operators in QEA

And now we give some useful definitions for understanding the QEA. We already know that a number of different representations can be used to encode the solutions onto individuals in Evolutionary Computation. The representations can be classified broadly as: binary, numeric, and symbolic [7].

QEA uses a new representation, called "Q-bit", for the probabilistic representation of the coexistence of the information 1 and 0. It is based on the physical concept of quantum bit. We also call "Q-individual" a string of Q-bits.

3.1 Q-bit

A Q-bit is defined as the smallest unit of information in QEA. Compared to bit, identified unambiguously by one "0" or by one "1", the Q-bit is defined with a pair of numbers (α, β), disposed as column vector $[\alpha\ \beta]^T$, where $|\alpha|^2 + |\beta|^2 = 1$. $|\alpha|^2$ gives the probability that the Q-bit will be found in the '0' state and $|\beta|^2$ gives the probability that the Q-bit will be found in the '1' state.

As previously said, a Q-bit may be in the "1" state, in the "0" state, or in a "hybrid" state, given from the linear superposition of the two states, properly "weighted" by factors α and β.

Using the "ket-notation", these three states can be described as:

$$|0\rangle = \begin{bmatrix} 1 \\ 0 \end{bmatrix}; \qquad |1\rangle = \begin{bmatrix} 0 \\ 1 \end{bmatrix}; \qquad |\varphi\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}.$$

In general, the algorithm processes Q-bit in the hybrid state. It just converges to "certain" values '0' or '1' at the end of the iterations.

3.2 Q-bit individual

Similarly to genetic one, where a generic individual of the population is represented by a string of bit, a Q-individual is represented by a string of Q-bit, defined as:

$$\begin{bmatrix} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \cdots & \beta_m \end{bmatrix}$$

such that $|\alpha_i|^2 + |\beta_i|^2 = 1$, with $i=1, 2, \ldots, m$.

For example, if there is a Q-individual composed by three Q-bits:

$$\begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & \frac{\sqrt{3}}{2} \end{bmatrix}$$

then, the states of the system can be represented as:

$$\frac{1}{4}|000\rangle + \frac{\sqrt{3}}{4}|001\rangle - \frac{1}{4}|010\rangle - \frac{\sqrt{3}}{4}|011\rangle + \frac{1}{4}|100\rangle + \frac{\sqrt{3}}{4}|101\rangle - \frac{1}{4}|110\rangle - \frac{\sqrt{3}}{4}|111\rangle$$

The above result means that the probabilities to represent the states $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$, and $|111\rangle$ are 1/16, 3/16, 1/16, 3/16, 1/16, 3/16, 1/16 and 3/16 respectively. By consequence, the Q-individual made by three-Q-bits contain the information of eight states. Evolutionary Computing using Q-bit has a better evidence of population diversity than other representations, because it can represents linear superposition of states in probabilistic way. So, only one Q-individual is enough to represent eight states, but in binary representation at least eight strings: (000), (001), (010), (011), (100), (101), (110) and (111) are needed.

3.3 Q-gate

A Q-gate is defined as a variation operator of QEA, whose functionality is to drive the individuals towards better solutions. The updated Q-bits should satisfy the normalization condition:

$$|\alpha'|^2 + |\beta'|^2 = 1$$

where α' *and* β' are the values of the updated Q-bit. An example of Q-gate is the following *rotation gate:*

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}$$

This operator, which provides a rotation of $\Delta\theta_i$ to the Q-bit, is the core of the algorithm and will be discussed in the next section.

4 Structure of algorithm

QEA works on one Q-individual, $Q(t) = \{q_1^t, q_2^t, \ldots, q_m^t\}$ at generation *t.*

The representation of *i-th* Q-bit q^t_i, with *i= 1, 2, ..., m,* where *m* is the number of Q-bit of the Q-individual, is defined as:

$$q_i^t = \begin{bmatrix} \alpha_i^t \\ \beta_i^t \end{bmatrix}$$

From *n* observations of each Q-bit of the Q-individual is built a population of bits, with size *n*.

The structure of the algorithm is described by the following figure.

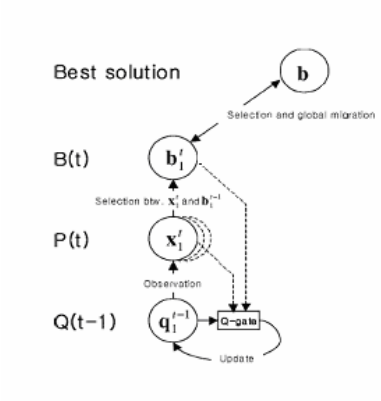

Figure 2: Overall structure of QEA

In the following, the algorithm QEA is shown in pseudo-code, reporting the particular operations performed on each step.

```
Procedure QEA
begin
      t ← 0
i)    initialize Q(t)
ii)   make P(t) by observing the states of Q(t)
iii)  evaluate P(t)
iv)   store the best solutions among P(t) into
      B(t)
      while (not termination-condition) do
      begin
         t ← t+1
v)       make P(t) by observing the states of
         Q(t-1)
vi)      evaluate P(t)
vii)     update Q(t) using Q-gates
viii)    store the best solutions among B(t-1)
         and P(t) into B(t)
ix)      store the best solution b among B(t)
x)       if (migration-condition)
         then migrate b or b_j^t to B(t) globally
         or locally, respectively
      end
end
```

```
Procedure Make (x)
begin
   i ← 0
   while (i < m) do
   begin
     i ← i + 1
     if random[0, 1) < |β_i|^2
     then x_i ← 1
     else x_i ← 0
   end
end
```

The rotation operator $U(\Delta\theta_i)$, applied at step vii), is the following:

$$U(\Delta\theta_i) = \begin{bmatrix} \cos(\Delta\theta_i) & -\sin(\Delta\theta_i) \\ \sin(\Delta\theta_i) & \cos(\Delta\theta_i) \end{bmatrix}$$

where $\Delta\theta_i$, with *i = 1, 2, ..., m is a rotation angle of each* Q-bit toward either 0 o 1 state depending on its sign. It allows the evolution of Q-bits, according to the polar plot depicted in figure 3.

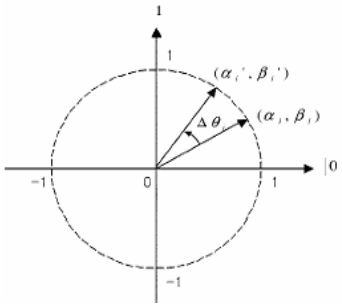

Figure 3: Polar plot of the rotation gate

5 Termination Criteria

To decide the appropriate termination of QEA, a proper termination condition is necessary. Although the maximum number of generations is a generally used termination criterion in EAs, in QEA the probability of the best solution can be employed as a termination criterion thanks to the probability representation. The termination condition is designed by using the probability of the best solution ***b*** as follows:

$$\Pr ob(b) = \prod_{i=1}^{m} p_{ji}$$

with

$$p_{ji} = \begin{cases} |\alpha_{ji}|^2, & if \quad b_i = 0 \\ |\beta_{ji}|^2, & if \quad b_i = 1 \end{cases}$$

where b_i is the *i*-th bit of the best solution b and (α_i, β_i) is the *i*-th Q-bit of the Q-individual.

The termination condition is defined as Prob(b) > γ_0 where $0 < \gamma_0 < 1$.

The probability Prob(b) represents the convergence of the Q-individual to the best solution. However, since the probability is sensitive to each Q-bit's probability, it is not easy to set the value γ_0: a slight difference of γ_0 can increase the processing time for a particular problem.

6 Hardware implementation

The quantum-evolutionary machine implementing the algorithm is made by a HW/SW platform, whose scheme is showed in figure 4.

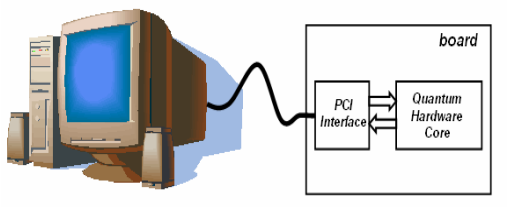

Figure 4: Platform HW/SW

6.1 Core organization

The core of the Quantum Hardware is shown in the scheme of figure 5. To describe the data population, a Dual Port SRAM memory has been used, while, to describe the Q-bits, a Single Port SRAM memory. In the system is also present a FIFO to handle data that must be send or receive from the PC. The core contains inside a state finite machine to handle the operations executed in hardware, Figure 6 shows a scheme of this machine.

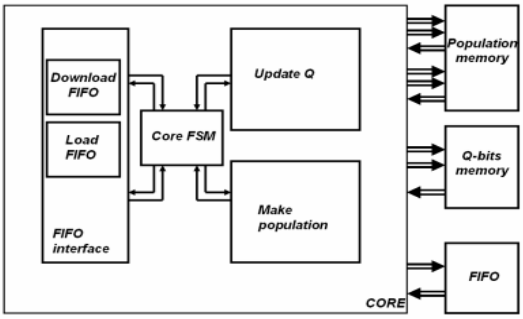

Figure 5: CORE Scheme

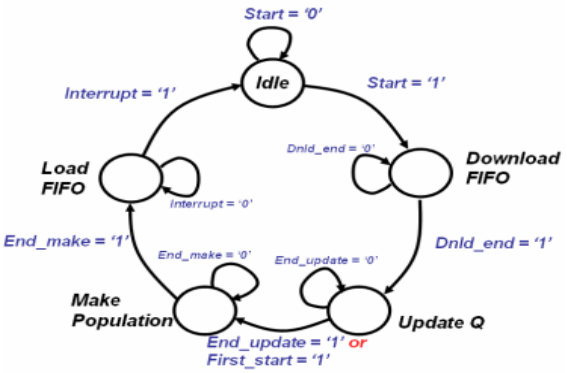

Figure 6: State Machine of the CORE

At every *start* of machine, all Q-bits in memory are initialized to $\left(\frac{1}{\sqrt{2}},\frac{1}{\sqrt{2}}\right)$. From the PC it can be send the *Start* and *First Start* signals; during the first execution of program the First Start signal is send and the machine begin whit the construction of first population (*Make_population*) based on the observation of the Q-bits inside the Q-individual at the value of initialization. Then, data are stored in the FIFO memory and the interrupt's pin is set to '1' to indicate to the PC that data are ready to be used.

The PC attends to compute the population fitness and the Best solution; then returns the data to the Hardware machine, arranging the FIFO memory consequently. So the PC set to '1' the value of Start and put it in *Stand by* for an interrupt. When Start is set to '1', the Core machine starts and handles the operations. The population data are downloaded from the FIFO and stored in the memory (Download FIFO). Then Q-bits are updated by the evaluation of the population fitness and the evaluation of the Best (Update_Q). A new pseudo - random population is built by the evaluation of the Q-bits updated in the Q-individual (Make_population). Finally, data are stored in FIFO (Load FIFO) and send to PC by the interrupt, again.

6.2 Memory representation of Q-bits

The population data is represented from 256 elements; every element is made by four variables (x, y, z, w). 31 bits have been used to represent each variable for the data and one bit for the evaluation (f(x, y, z, t) > b(x, y, z, w, t – 1)). So, to easily represent the population data in hardware, it has been used a memory of 256 words of 128 bits each one. The data was stored as shown in figure 7a. A similar strategy has been used for representing Q-bits in hardware. Every Q-bit is defined as the ordinate couple (α,β) so that $|\alpha|^2+|\beta|^2=1$ and $\alpha^2,\beta^2\le 1$.

To perform efficiently the Q-bits operations, the ordinate couple (α,β) is built by a couple of fixed-point values of 16 bits. So, the Q-bits are stored in a memory with 124 words, (in fact 124 bit are necessary to represent a population data (x, y, z, w)) of 32 bits (31+1), as showed in figure 7b.

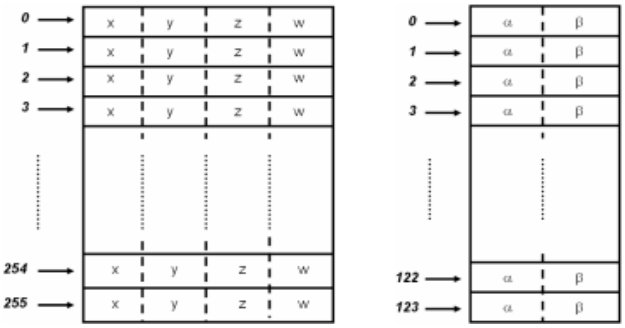

Figure 7a-7b: Data organization and Q-bits in memory

6.3 Update Q-bits

The updates of Q-bits are made by a state machine, which scheme is shown in Figure 8. The state machine also performs the computing operations concerning 'angle rotation' and 'rotation of Q-bit'.

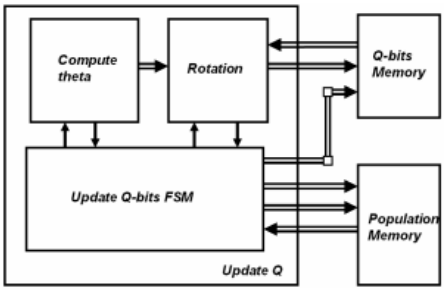

Figure. 8: Update scheme of Q-bits

The state machine that handles the Q-bits update performs a high number of operations. In fact for each Q-bit (ex. Q(j)) it's necessary:
1. to take the *i-element* from the population;
2. to extract the element *p(i,j)*;
3. to update the angle rotation *theta*.

Only when the contribution of all elements of a population was computed (i = 0, …,255), we can update the Q-bit and to begin the evaluation of a new Q-bit (ex. Q(j + 1)).

Once all Q-bits were updated, the *End* signal is set to '1', the machine goes to idle state and the Quantum Evolutionary Algorithm goes to *Make_population* phase. Figure 9 shows the scheme of the state machine that handles the update of the Q-bits.

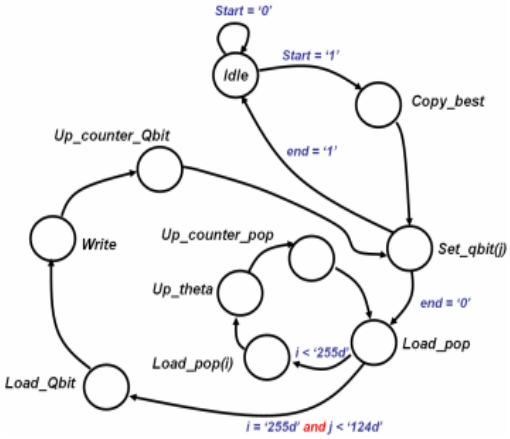

Figure 9: Update of Q-bits FSM

The compute of *theta* is performed by the *Compute_theta* module, during the phase of *Up_theta*. To realize this computation, it was designed a structure with a multiplexer and an accumulator, as shown in figure 10.

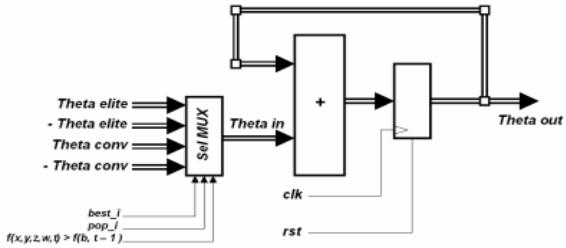

Figure.10: Compute_theta

Q-bits rotation is done by the *Rotation* module. Here it is shown the formula to perform the rotation of Q-bits.

$$\left(\alpha^{'},\beta^{'}\right)=\begin{pmatrix}\cos(\vartheta) & -\sin(\vartheta)\\ \sin(\vartheta) & \cos(\vartheta)\end{pmatrix}\begin{pmatrix}\alpha\\ \beta\end{pmatrix}$$

In Figure 11 is described the circuit for the Rotation module which was employed for the rotation of Q-bits.

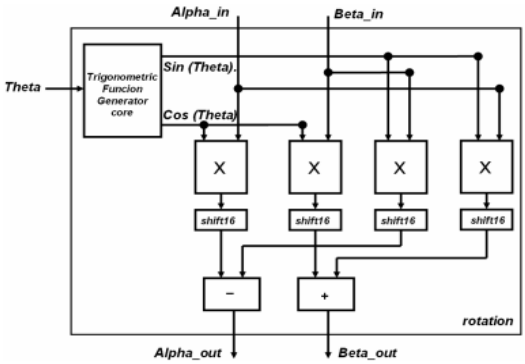

Figure. 11: Rotation Module

6.4 Make Population

The generation of the population is performed by a machine that develops pseudo-random number, starting from evaluation of Q-bits states (see Procedure *Make* in section 4). In figure 12 is shown a scheme of *Make_population* module. Figure 13 shows the scheme of state machine that handles the operations.

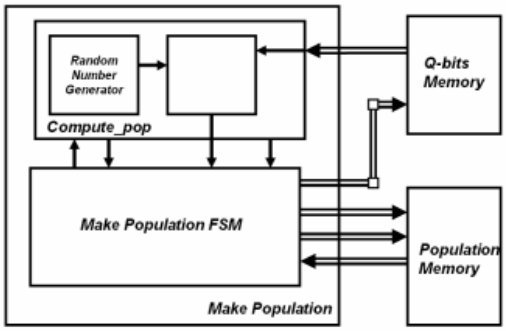

Figure 12: Make_population scheme

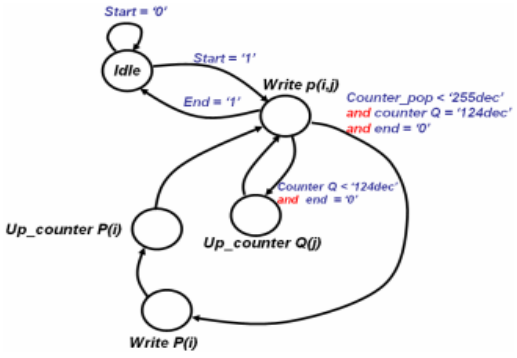

Figure 13: Make_population Finite State Machine

When the state machine starts, by the *Start* signal, the Q-bit(0) is load from Q-bits memory. Evaluation by the Q-bit(0) produces the element p(0,0), that represents the first bit of the first element of population P(0). Then the counter of Q-bits is increased, it is red Q-bit(1), and is produced p(0,1) that represents the second bit of the first population element P(0). So the procedure goes on in this way until to the generation of all bits in the first population individual. When P(0) is generated, it is written in the Population Memory, counter P is increased and counter Q is reset. So it is performed the generation of P(1), P(2) until P(255). When the whole population is generated, the state machine sets to '1' the end signal and puts again itself in waiting for the *start* signal.

The generation of the population is pseudo-random: the Q-bit represents probability that the new element p(i,j) generated is '1' or '0'. In figure 14 is shown the *Compute_pop* module, that performs the generation of element p(i,j) from the evaluation of Q-bit(j).

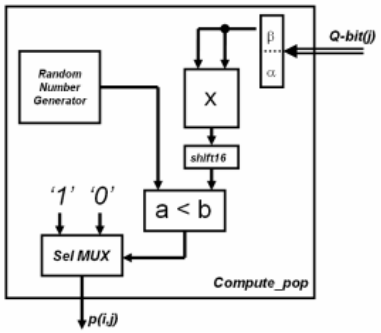

Figure. 14: Compute_pop scheme module

To generate a sequence of random numbers, it was used a module that implements the algorithm:

$$x_{n+1} = P_1(n,t) * x_n + P_2(n,t)$$

where P_1 and P_2 represent pseudo-random values produced by two particular counters that freely run during all the algorithm execution.

References:

[1] D. E. Goldberg: "Genetic Algorithms in Search, Optimization and Machine Learning". Addison Wesley, 1989. ISBN: 0-20115-767-5

[2] Lawrence Davis Editor "Handbook of Genetic Algorithms" Van Nostrand Reinhold Computer Library, 1991. ISBN: 1-85032-825-0

[3] Kuk-Hyun Han and Jong-Hwan Kim, "Quantum-Inspired Evolutionary Algorithm for a Class of Combinatorial Optimization" IEEE transactions on evolutionary computation, vol. 6, no. 6, December 2002

[4] Kuk-Hyun Han and Jong-Hwan Kim "Quantum-Inspired Evolutionary Algorithms With a New Termination Criterion, H_ϵ Gate,and Two-Phase Scheme" IEEE transactions on evolutionary computation, vol. 8, no. 2, April 2004

[5] K.-H. Han and J.-H. Kim, "Genetic quantum algorithm and its application to combinatorial optimization problem," in *Proc. 2000 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press, July 2000, vol. 2, pp. 1354–1360.

[6] K.-H. Han, K.-H. Park, C.-H. Lee, and J.-H. Kim, "Parallel quantum-inspired genetic algorithm for combinatorial optimization problem," in *Proc. 2001 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press, May 2001, vol. 2, pp. 1422–1429.

[7] R. Hinterding, "Representation, constraint satisfaction and the knapsack problem," in *Proc. 1999 Congress on Evolutionary Computation*. Piscataway, NJ: IEEE Press, July 1999, vol. 2, pp. 1286–1292.