Genetic Quantum Algorithm and its Application to Combinatorial Optimization Problem

Kuk-Hyun Han
Dept. of Electrical Engineering, KAIST,
373-1, Kusong-dong Yusong-gu
Taejon, 305-701, Republic of Korea
khhan@vivaldi.kaist.ac.kr

Jong-Hwan Kim
Dept. of Electrical Engineering, KAIST,
373-1, Kusong-dong Yusong-gu
Taejon, 305-701, Republic of Korea
johkim@vivaldi.kaist.ac.kr

Abstract- This paper proposes a novel evolutionary computing method called a genetic quantum algorithm (GQA). GQA is based on the concept and principles of quantum computing such as qubits and superposition of states. Instead of binary, numeric, or symbolic representation, by adopting qubit chromosome as a representation GQA can represent a linear superposition of solutions due to its probabilistic representation. As genetic operators, quantum gates are employed for the search of the best solution. Rapid convergence and good global search capability characterize the performance of GQA. The effectiveness and the applicability of GQA are demonstrated by experimental results on the knapsack problem, which is a well-known combinatorial optimization problem. The results show that GQA is superior to other genetic algorithms using penalty functions, repair methods, and decoders.

1 Introduction

Many efforts on quantum computers have progressed actively since the early 1990's because these computers were shown to be more powerful than classical computers on various specialized problems. But if there is no quantum algorithm that solves practical problems, quantum computer hardware may be useless. It could be considered as a computer without operating system.

Although there would be significant benefit from new quantum algorithms that could solve computational problems faster than classical algorithms, to date, only a few quantum algorithms are known. Nevertheless, quantum computing is attracting serious attention, since its superiority was demonstrated by a few quantum algorithms such as Shor's quantum factoring algorithm [1, 2] and Grover's database search algorithm [3, 4]. Shor's algorithm finds the prime factors of an n-digit number in polynomial-time, while the best-known classical factoring algorithms require time $O\left(2^{n^{\frac{1}{3}}\log(n)^{\frac{2}{3}}}\right)$. Grover's database search algorithm can find an item in an unsorted list of n items in $O\left(\sqrt{n}\right)$ steps, while classical algorithms require $O(n)$.

Research on merging evolutionary computing and quantum computing has been started by some researchers since late 1990's. They can be classified into two fields. One concentrates on generating new quantum algorithms using automatic programming techniques such as genetic programming [5]. The absence of new quantum algorithms motivated this work. The other concentrates on quantum-inspired evolutionary computing for a classical computer [6], a branch of study on evolutionary computing that is characterized by certain principles of quantum mechanics such as standing waves, interference, coherence, etc.

This paper offers a novel evolutionary computing algorithm called a genetic quantum algorithm (GQA). GQA is characterized by principles of quantum computing including concepts of qubits and superposition of states. GQA uses a qubit representation instead of binary, numeric, or symbolic representations. GQA can imitate parallel computation in classical computers.

This paper is organized as follows. Section 2 describes a novel evolutionary computing algorithm, GQA. Section 3 contains a description of the experiment with GAs and GQAs for knapsack problems for comparison purpose. Section 4 summarizes and analyzes the experimental results. Concluding remarks follow in Section 5.

2 Genetic Quantum Algorithm (GQA)

GQA is based on the concepts of qubits and superposition of states of quantum mechanics. The smallest unit of information stored in a two-state quantum computer is called a quantum bit or qubit [7]. A qubit may be in the '1' state, in the '0' state, or in any superposition of the two. The state of a qubit can be represented as

$$|\Psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{1}$$

where α and β are complex numbers that specify the probability amplitudes of the corresponding states. $|\alpha|^2$ gives the probability that the qubit will be found in '0' state and $|\beta|^2$ gives the probability that the qubit will be found in the '1' state. Normalization of the state to unity guarantees

$$|\alpha|^2 + |\beta|^2 = 1. \tag{2}$$

If there is a system of m-qubits, the system can represent 2^m states at the same time. However, in the act of observing a quantum state, it collapses to a single state [8].

2.1 Representation

It is possible to use a number of different representations to encode the solutions onto chromosomes in evolutionary computation. The classical representations can be broadly classified as: binary, numeric, and symbolic [9]. GQA uses a novel representation that is based on the concept of qubits. One qubit is defined with a pair of complex numbers, (α, β), as

$$\begin{bmatrix} \alpha \\ \beta \end{bmatrix},$$

which is characterized by (1) and (2). And an m-qubits representation is defined as

$$\left[\begin{array}{c|c|c|c} \alpha_1 & \alpha_2 & \cdots & \alpha_m \\ \beta_1 & \beta_2 & \cdots & \beta_m \end{array}\right], \tag{3}$$

where $|\alpha_i|^2 + |\beta_i|^2 = 1, i = 1, 2, \cdots, m$. This representation has the advantage that it is able to represent any superposition of states. If there is, for instance, a three-qubits system with three pairs of amplitudes such as

$$\left[\begin{array}{c|c|c} \frac{1}{\sqrt{2}} & 1.0 & \frac{1}{2} \\ \frac{1}{\sqrt{2}} & 0.0 & \frac{\sqrt{3}}{2} \end{array}\right], \tag{4}$$

the state of the system can be represented as

$$\frac{1}{2\sqrt{2}}|000\rangle + \frac{\sqrt{3}}{2\sqrt{2}}|001\rangle + \frac{1}{2\sqrt{2}}|100\rangle + \frac{\sqrt{3}}{2\sqrt{2}}|101\rangle. \tag{5}$$

The above result means that the probabilities to represent the state $|000\rangle$, $|001\rangle$, $|100\rangle$, and $|101\rangle$ are $\frac{1}{8}$, $\frac{3}{8}$, $\frac{1}{8}$, and $\frac{3}{8}$, respectively. By consequence, the three-qubits system of (4) has four states information at the same time.

Evolutionary computing with the qubit representation has a better characteristic of diversity than classical approaches, since it can represent superposition of states. Only one qubit chromosome such as (4) is enough to represent four states, but in classical representation at least four chromosomes, (000), (001), (100), and (101) are needed. Convergence can be also obtained with the qubit representation. As $|\alpha_i|^2$ or $|\beta_i|^2$ approaches to 1 or 0, the qubit chromosome converges to a single state and the property of diversity disappears gradually. That is, the qubit representation is able to possess the two characteristics of exploration and exploitation, simultaneously.

2.2 GQA

The structure of GQA is described in the following.

procedure GQA
begin
  $t \leftarrow 0$
  initialize $Q(t)$
  make $P(t)$ by observing $Q(t)$ states
  evaluate $P(t)$
  store the best solution among $P(t)$
  while (**not** termination-condition) **do**
  begin
    $t \leftarrow t + 1$
    make $P(t)$ by observing $Q(t-1)$ states
    evaluate $P(t)$
    update $Q(t)$ using quantum gates $U(t)$
    store the best solution among $P(t)$
  end
end

GQA is a probabilistic algorithm which is similar to a genetic algorithm. GQA maintains a population of qubit chromosomes, $Q(t) = \{\mathbf{q}_1^t, \mathbf{q}_2^t, \cdots, \mathbf{q}_n^t\}$ at generation t, where n is the size of population, and $\mathbf{q}_j^t$ is a qubit chromosome defined as

$$\mathbf{q}_j^t = \left[\begin{array}{c|c|c|c} \alpha_1^t & \alpha_2^t & \cdots & \alpha_m^t \\ \beta_1^t & \beta_2^t & \cdots & \beta_m^t \end{array}\right], \tag{6}$$

where m is the number of qubits, i.e., the string length of the qubit chromosome, and $j = 1, 2, \cdots, n$.

In the step of 'initialize $Q(t)$,' α_i^t and β_i^t, $i = 1, 2, \cdots, m$, of all $\mathbf{q}_j^t$, $j = 1, 2, \cdots, n$, in $Q(t)$ are initialized with $\frac{1}{\sqrt{2}}$. It means that one qubit chromosome, $\mathbf{q}_j^t|_{t=0}$ represents the linear superposition of all possible states with the same probability:

$$|\Psi_{\mathbf{q}_j^0}\rangle = \sum_{k=1}^{2^m} \frac{1}{\sqrt{2^m}}|S_k\rangle,$$

where S_k is the k-th state represented by the binary string $(x_1 x_2 \cdots x_m)$, where x_i, $i = 1, 2, \cdots, m$, is either 0 or 1. The next step makes a set of binary solutions, $P(t)$, by observing $Q(t)$ states, where $P(t) = \{\mathbf{x}_1^t, \mathbf{x}_2^t, \cdots, \mathbf{x}_n^t\}$ at generation t. One binary solution, $\mathbf{x}_j^t$, $j = 1, 2, \cdots, n$, is a binary string of the length m, and is formed by selecting each bit using the probability of qubit, either $|\alpha_i^t|^2$ or $|\beta_i^t|^2$, $i = 1, 2, \cdots, m$, of $\mathbf{q}_j^t$. Each solution $\mathbf{x}_j^t$ is evaluated to give some measure of its fitness. The initial best solution is then selected and stored among the binary solutions, $P(t)$.

In the **while** loop, one more step, 'update $Q(t)$,' is included to have fitter states of the qubit chromosomes. A set of binary solutions, $P(t)$, is formed by observing $Q(t-1)$ states as with the procedure described before, and each binary solution is evaluated to give the fitness value. In the next step, 'update $Q(t)$,' a set of qubit chromosomes $Q(t)$ is updated by applying some appropriate quantum gates [1] $U(t)$, which is formed by using the binary solutions $P(t)$ and the stored best solution. The appropriate quantum gates can be designed in compliance with practical problems. Rotation gates, for instance, will be used for knapsack problems in the next sec-

[1] Quantum gates are reversible gates and can be represented as unitary operators acting on the qubit basis states: $U^\dagger U = UU^\dagger$, where $U^\dagger$ is the hermitian adjoint of U. There are several quantum gates, such as NOT gate, Controlled NOT gate, Rotation gate, Hadamard gate, etc.[7].

tion, such as

$$U(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad (7)$$

where θ is a rotation angle. This step makes the qubit chromosomes converge to the fitter states. The best solution among $P(t)$ is selected in the next step, and if the solution is fitter than the stored best solution, the stored solution is changed by the new one. The binary solutions $P(t)$ are discarded at the end of the loop.

It should be noted that some genetic operators can be applied, such as mutation which creates new individuals by a small change in a single individual, and crossover which creates new individuals by combining parts from two or more individuals. Mutation and crossover can make the probability of linear superposition of states change. But as GQA has diversity caused by the qubit representation, there is no need to use the genetic operators. If the probabilities of mutation and crossover are high, the performance of GQA can be decreased notably.

In GQA, the population size, i.e., the number of qubit chromosomes is kept the same all the time. This is caused by conservation of qubits based on quantum computing. GQA with the qubit representation can have better convergence with diversity than conventional GAs which have fixed 0 and 1 information.

3 Experiment

The knapsack problem, a kind of combinatorial optimization problem, is used to investigate the performance of GQA. The knapsack problem can be described as selecting from among various items those items which are most profitable, given that the knapsack has limited capacity. The 0-1 knapsack problem is described as: given a set of m items and a knapsack, select a subset of the items so as to maximize the profit $f(\mathbf{x})$:

$$f(\mathbf{x}) = \sum_{i=1}^{m} p_i x_i,$$

subject to

$$\sum_{i=1}^{m} w_i x_i \leq C,$$

where $\mathbf{x} = (x_1 \cdots x_m)$, x_i is 0 or 1, p_i is the profit of item i, w_i is the weight of item i, and C is the capacity of the knapsack.

In this section, some conventional GA methods are described to experiment with the 0-1 knapsack problem, and the detailed algorithm of GQA for the knapsack problem follows.

3.1 Conventional GA methods

Three types of conventional algorithms are described and tested: algorithms based on penalty functions, algorithms based on repair methods, and algorithms based on decoders [10].

In all algorithms based on penalty functions, a binary string of the length m represents a chromosome $\mathbf{x}$ to the problem. The profit $f(\mathbf{x})$ of each string is determined as

$$f(\mathbf{x}) = \sum_{i=1}^{m} p_i x_i - Pen(\mathbf{x}),$$

where $Pen(\mathbf{x})$ is a penalty function. There are many possible strategies for assigning the penalty function [11, 12]. Three types of penalties are considered, such as logarithmic penalty, linear penalty, and quadratic penalty:

$$\begin{aligned} Pen_1(\mathbf{x}) &= \log_2\left(1 + \rho\left(\textstyle\sum_{i=1}^{m} w_i x_i - C\right)\right), \\ Pen_2(\mathbf{x}) &= \rho\left(\textstyle\sum_{i=1}^{m} w_i x_i - C\right), \\ Pen_3(\mathbf{x}) &= \left(\rho\left(\textstyle\sum_{i=1}^{m} w_i x_i - C\right)\right)^2, \end{aligned}$$

where ρ is $\max_{i=1\cdots m}\{p_i/w_i\}$.

In algorithms based on repair methods, the profit $f(\mathbf{x})$ of each string is determined as

$$f(\mathbf{x}) = \sum_{i=1}^{m} p_i x'_i,$$

where $\mathbf{x}'$ is a repaired vector of the original vector $\mathbf{x}$. Original chromosomes are replaced with a 5% probability in the experiment. The two repair algorithms considered here differ only in selection procedure, which chooses an item for removal from the knapsack:

Rep_1 (random repair): The selection procedure selects a random element from the knapsack.
Rep_2 (greedy repair): All items in the knapsack are sorted in the decreasing order of their profit to weight ratios. The selection procedure always chooses the last item for deletion.

A possible decoder for the knapsack problem is based on an integer representation. Each chromosome is a vector of m integers; the i-th component of the vector is an integer in the range from 1 to $m - i + 1$. The ordinal representation references a list L of items; a vector is decoded by selecting appropriate item from the current list. The two algorithms based on decoders considered here differ only in the procedure of building a list L of items:

Dec_1 (random decoding): The build procedure creates a list L of items such that the order of items on the list corresponds to the order of items in the input file which is random.
Dec_2 (greedy decoding): The build procedure creates a list L of items in the decreasing order of their profit to weight ratios.

3.2 GQA for the knapsack problem

The algorithm of GQA for the knapsack problem is based on the structure of GQA proposed and it contains a repair

algorithm. The algorithm can be written as follows:

procedure GQA
begin
  $t \leftarrow 0$
  initialize $Q(t)$
  make $P(t)$ by observing $Q(t)$ states
  repair $P(t)$
  evaluate $P(t)$
  store the best solution $\mathbf{b}$ among $P(t)$
  while ($t < MAX\_GEN$) **do**
  begin
    $t \leftarrow t + 1$
    make $P(t)$ by observing $Q(t-1)$ states
    repair $P(t)$
    evaluate $P(t)$
    update $Q(t)$
    store the best solution $\mathbf{b}$ among $P(t)$
  end
end

A qubit string of the length m represents a linear superposition of solutions to the problem as in (6). The length of a qubit string is the same as the number of items. The i-th item can be selected for the knapsack with probability $|\beta_i|^2$ or $(1-|\alpha_i|^2)$. Thus, a binary string of the length m is formed from the qubit string. For every bit in the binary string, we generate a random number r from the range $[0..1]$; if $r > |\alpha_i|^2$, we set the bit of the binary string. The binary string $\mathbf{x}_j^t$, $j = 1, 2, \cdots, n$, of $P(t)$ represents a j-th solution to the problem. For notational simplicity, $\mathbf{x}$ is used instead of $\mathbf{x}_j^t$ in the following. The i-th item is selected for the knapsack iff $x_i = 1$, where x_i is the i-th bit of $\mathbf{x}$. The binary string $\mathbf{x}$ is determined as follows:

procedure make (x)
begin
  $i \leftarrow 0$
  while ($i < m$) **do**
  begin
    $i \leftarrow i + 1$
    if $random[0, 1) > |\alpha_i|^2$
    then $x_i \leftarrow 1$
    else $x_i \leftarrow 0$
  end
end

The repair algorithm of GQA for the knapsack problem is implemented as follows:

procedure repair (x)
begin
  knapsack-overfilled $\leftarrow$ false
  if $\sum_{i=1}^{m} w_i x_i > C$
  then knapsack-overfilled $\leftarrow$ true
  while (knapsack-overfilled) **do**
  begin
    select an i-th item from the knapsack
    $x_i \leftarrow 0$
    if $\sum_{i=1}^{m} w_i x_i \leq C$
    then knapsack-overfilled $\leftarrow$ false
  end
  while (**not** knapsack-overfilled) **do**
  begin
    select a j-th item from the knapsack
    $x_j \leftarrow 1$
    if $\sum_{i=1}^{m} w_i x_i > C$
    then knapsack-overfilled $\leftarrow$ true
  end
  $x_j \leftarrow 0$
end

x_i	b_i	$f(\mathbf{x}) \geq f(\mathbf{b})$	$\Delta\theta_i$	$s(\alpha_i\beta_i)$			
				$\alpha_i\beta_i > 0$	$\alpha_i\beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	$false$	0	0	0	0	0
0	0	$true$	0	0	0	0	0
0	1	$false$	0	0	0	0	0
0	1	$true$	0.05π	-1	$+1$	± 1	0
1	0	$false$	0.01π	-1	$+1$	± 1	0
1	0	$true$	0.025π	$+1$	-1	0	± 1
1	1	$false$	0.005π	$+1$	-1	0	± 1
1	1	$true$	0.025π	$+1$	-1	0	± 1

Table 1: Lookup table of θ_i, where $f(\cdot)$ is the profit, $s(\alpha_i\beta_i)$ is the sign of θ_i, and b_i and x_i are the i-th bits of the best solution $\mathbf{b}$ and the binary solution $\mathbf{x}$, respectively.

The profit of a binary solution $\mathbf{x}$ is evaluated by $\sum_{i=1}^{m} p_i x_i$, and it is used to find the best solution $\mathbf{b}$ after the update of $\mathbf{q}_j$, $j = 1, 2, \cdots, n$. A qubit chromosome $\mathbf{q}_j$ is updated by using the rotation gate $U(\theta)$ of (7) in this algorithm. The i-th qubit value (α_i, β_i) is updated as

$$\begin{bmatrix} \alpha_i' \\ \beta_i' \end{bmatrix} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{bmatrix} \begin{bmatrix} \alpha_i \\ \beta_i \end{bmatrix}. \quad (8)$$

In this knapsack problem θ_i is given as $s(\alpha_i\beta_i)\Delta\theta_i$. The parameters used are shown in Table 1. For example, if the condition, $f(\mathbf{x}) \geq f(\mathbf{b})$, is satisfied and x_i and b_i are 1 and 0, respectively, we can set the value of $\Delta\theta_i$ as 0.025π and $s(\alpha_i\beta_i)$ as $+1$, -1, or 0 according to the condition of $\alpha_i\beta_i$ so as to increase the probability of the state $|1\rangle$. The value of $\Delta\theta_i$ has an effect on the speed of convergence, but if it is too big, the solutions may diverge or have a premature convergence to a local optimum. The sign $s(\alpha_i\beta_i)$ determines the direction of convergence to a global optimum. The lookup table can be used as a strategy for convergence. This **update** procedure can be described as follows:

procedure update (q)
begin
  $i \leftarrow 0$

♯ of items			CGAs								GQAs	
			$Pen1$	$Pen2$	$Pen3$	$Rep1$	$Rep2$	$Dec1$	$Dec2$	$P2+R1$	$GQA(1)$	$GQA(10)$
100	profits	b.	557.7	581.4	566.0	561.1	560.2	514.7	511.0	582.2	597.5	612.5
		m.	545.4	569.7	556.1	546.5	546.3	503.9	500.0	571.1	583.7	603.9
		w.	535.1	562.6	551.1	537.3	536.6	496.3	493.3	562.3	562.5	592.7
	$t(sec/run)$		1.329	1.333	1.323	1.142	1.151	3.510	10.51	1.360	0.054	0.382
250	profits	b.	-	-	-	-	-	-	-	1391.9	1444.9	1480.3
		m.	-	-	-	-	-	-	-	1382.1	1412.4	1467.1
		w.	-	-	-	-	-	-	-	1364.8	1385.8	1443.8
	$t(sec/run)$		-	-	-	-	-	-	-	3.292	0.141	1.380
500	profits	b.	-	-	-	-	-	-	-	2744.2	2824.1	2860.0
		m.	-	-	-	-	-	-	-	2720.8	2771.5	2841.3
		w.	-	-	-	-	-	-	-	2699.2	2744.3	2812.5
	$t(sec/run)$		-	-	-	-	-	-	-	6.532	0.324	3.994

Table 2: Experimental results of the knapsack problem: the maximum number of generations 500, the number of runs 25. $P2+R1$ means the algorithm implemented by $Pen2$ and $Rep1$, and $b.$, $m.$, and $w.$ means $best$, $mean$, and $worst$, respectively. $t(sec/run)$ represents the elapsed time per one run, and '-' means that an experiment did not made in this case.

while $(i < m)$ **do**
begin
$i \leftarrow i + 1$
determine θ_i with the lookup table
obtain (α'_i, β'_i) as:
$[\alpha'_i \ \beta'_i]^T = U(\theta_i)\,[\alpha_i \ \beta_i]^T$
end
$\mathbf{q} \leftarrow \mathbf{q}'$
end

The **update** procedure can be implemented in various methods with appropriate quantum gates. It depends on a given problem.

4 Results

In all experiments strongly correlated sets of data were considered:

$$\begin{aligned} w_i &= uniformly\ random[1, 10) \\ p_i &= w_i + 5, \end{aligned}$$

the average knapsack capacity was used:

$$C = \frac{1}{2}\sum_{i=1}^{m} w_i,$$

and the data files were unsorted. The population size of the eight conventional genetic algorithms (CGAs) was equal to 100. Probabilities of crossover and mutation were fixed: 0.65 and 0.05, respectively, as in [10]. The population size of GQA(1) was equal to 1, and the population size of GQA(10) was equal to 10, this being the only difference between GQA(1) and GQA(10). As a performance measure of the algorithm we collected the best solution found within 500 generations over 25 runs, and we checked the elapsed time per one run. A Pentium-III $500MHz$ was used, running Visual C++ 6.0.

Table 2 shows the experimental results of the knapsack problems with 100, 250, and 500 items. In the case of 100 items, GQA yielded superior results as compared to all the other CGAs. The CGA designed by using a linear penalty function and random repair algorithm outperformed all other CGAs, but is behind GQA(1) as well as GQA(10) in performance. The results show that GQA performs well in spite of small size of population. Judging from the results, GQA can search solutions near the optimum within a short time as compared to CGAs. In the cases of 250 and 500 items, the CGA that outperforms all the other CGAs was tested for comparison purpose with GQA. The experimental results again demonstrate the superiority of GQA.

Figure 1 shows the progress of the mean of best profits and the mean of average profits of population found by GQA(1), GQA(10), and CGA over 25 runs for 100, 250, and 500 items. GQA performs better than CGA in terms of convergence rate and final results. In the beginning of the plotting of the best profits, GQA(1) shows a slower convergence rate than GQA(10) and CGA due to its small population number. After 50 generations, GQA(10) and GQA(1) maintain a nearly constant convergence rate, while CGA's convergence rate reduces substantially. After 200 generations, even though convergence rate of GQA reduces, GQAs show a faster convergence rate than CGA due to its better global search ability. GQAs' final results are larger than CGA's in 1000 generations. The tendency of convergence rate can be shown clearly in the results of the mean of average profits of population. In the beginning, convergence rates of all the algorithms increase. But CGA maintains a nearly constant profit due to its premature convergence immediately, while GQA approaches towards the neighborhood of global optima with a constant

convergence rate. GQAs display no premature convergence which is a common problem of CGAs until 1000 generations.

The experimental results demonstrate the effectiveness and the applicability of GQA. Especially, Figure 1 shows the excellent global search ability and the superiority of convergence ability of GQA.

5 Conclusions

This paper proposed a novel evolutionary computing algorithm, GQA with a quantum representation. GQA is based on the principles of quantum computing such as concepts of qubits and superposition of states. GQA can represent a linear superposition of states, and there is no need to include many individuals. GQA has an excellent ability of global search due to its diversity caused by the probabilistic representation, and it can approach better solutions than CGA's in a short time. The knapsack problem, a kind of combinatorial optimization problems, is used to discuss the performance of GQA. It was showed that GQA's convergence and global search ability are superior to CGA's. The experimental results demonstrate the effectiveness and the applicability of GQA.

References

[1] P. W. Shor, "Quantum Computing," *Documenta Mathematica*, vol. Extra Volume ICM, pp. 467-486, 1998, http://east.camel.math.ca/ EMIS/ journals/ DMJDMV/ xvol-icm/ 00/ Shor.MAN.html.

[2] P. W. Shor, "Algorithms for Quantum Computation: Discrete Logarithms and Factoring," in *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, pp. 124-134, 1994.

[3] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the 28th ACM Symposium on Theory of Computing*, pp. 212-219, 1996.

[4] L. K. Grover, "Quantum Mechanical Searching," in *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 2255-2261, Jul 1999.

[5] L. Spector, H. Barnum, H. J. Bernstein and N. Swamy, "Finding a Better-than-Classical Quantum AND/OR Algorithm using Genetic Programming," in *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 2239-2246, Jul 1999.

[6] A. Narayanan and M. Moore, "Quantum-inspired genetic algorithms," in *Proceedings of IEEE International Conference on Evolutionary Computation*, pp. 61-66, 1996.

[7] T. Hey, "Quantum computing: an introduction," *Computing & Control Engineering Journal*, pp. 105-112, Jun 1999.

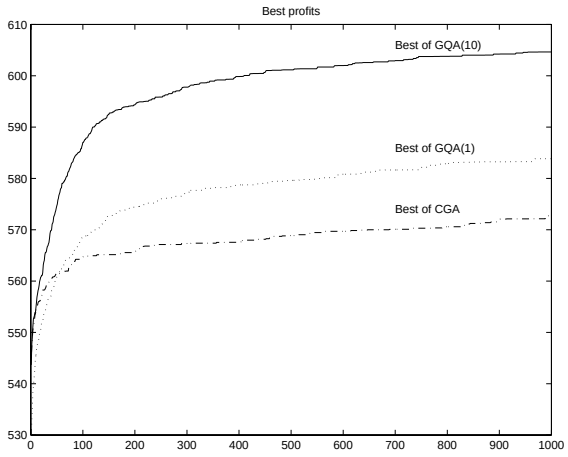
[8] A. Narayanan, "Quantum computing for beginners," in *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 2231-2238, Jul 1999.

[9] R. Hinterding, "Representation, Constraint Satisfaction and the Knapsack Problem," in *Proceedings of the 1999 Congress on Evolutionary Computation*, pp. 1286-1292, Jul 1999.

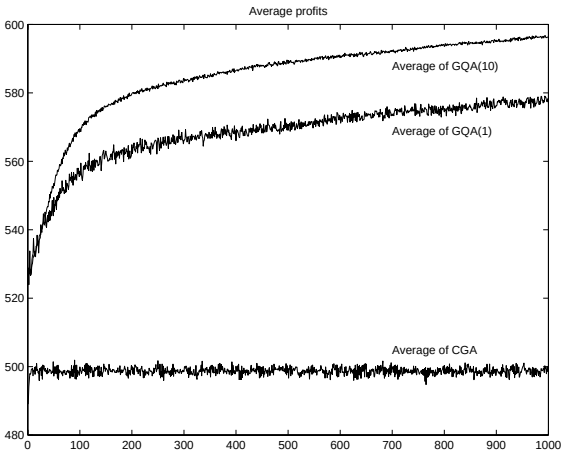
[10] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 3rd, revised and extended edition, 1999.

[11] J.-H. Kim and H. Myung, "Evolutionary Programming Techniques for Constrained Optimization Problems," *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 2, pp. 129-140, Jul 1997.

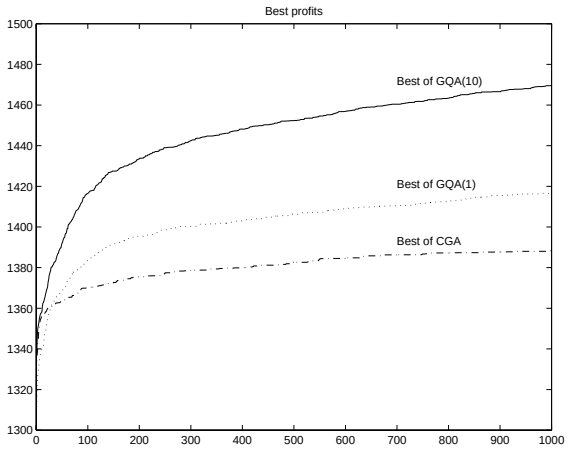
[12] X. Yao, *Evolutionary Computation: Theory and Applications*, World Scientific, Singapore, 1999.



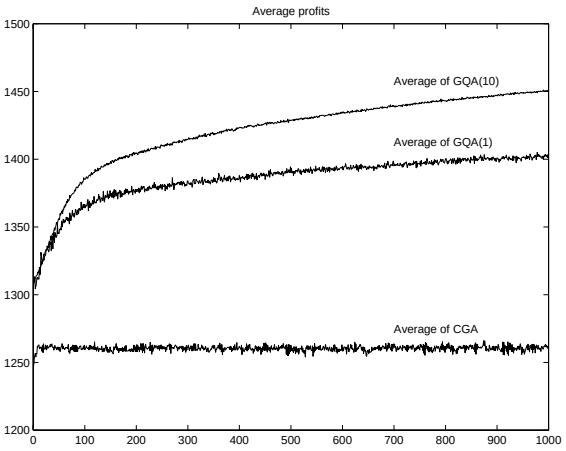
(a) best profits (100 items)



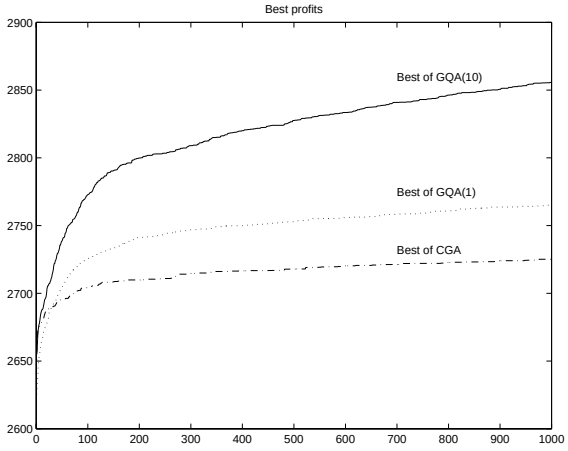
(b) average profits (100 items)



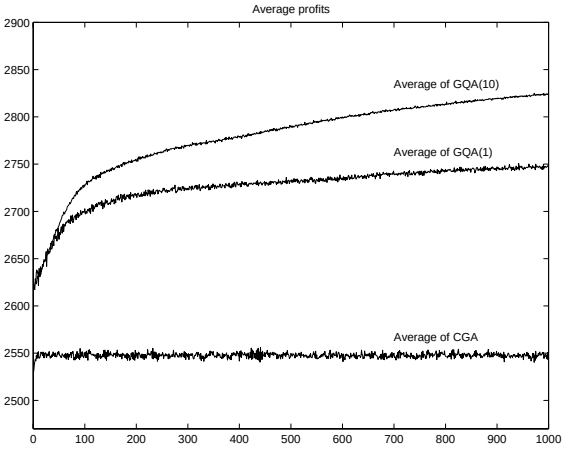
(c) best profits (250 items)



(d) average profits (250 items)



(e) best profits (500 items)



(f) average profits (500 items)

Figure 1: Comparison between CGA and GQA on the knapsack problem. The vertical axis is the profit value of knapsack, and the horizontal axis is the number of generations. (a), (c), (e) show the best profits, and (b), (d), (f) show the average profits. Both were averaged over 25 runs.