Parallel Quantum-inspired Genetic Algorithm for Combinatorial Optimization Problem

Kuk-Hyun Han **Kui-Hong Park** **Chi-Ho Lee** **Jong-Hwan Kim**

Dept. of Electrical Engineering and Computer Science,
Korea Advanced Institute of Science and Technology(KAIST),
373-1, Kusong-dong Yusong-gu, Taejon, 305-701, Republic of Korea
Email: {khhan,khpark,chiho,johkim}@vivaldi.kaist.ac.kr

Abstract- This paper proposes a new parallel evolutionary algorithm called parallel quantum-inspired genetic algorithm (PQGA). Quantum-inspired genetic algorithm(QGA) 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, QGA can represent a linear superposition of solutions due to its probabilistic representation. QGA is suitable for parallel structure because of rapid convergence and good global search capability. That is, QGA is able to possess the two characteristics of exploration and exploitation, simultaneously. The effectiveness and the applicability of PQGA are demonstrated by experimental results on the knapsack problem, which is a well-known combinatorial optimization problem. The results show that PQGA is superior to QGA as well as other conventional genetic algorithms.

1 Introduction

Evolutionary algorithms(EAs) are principally stochastic searches and optimization methods based on the principles of natural biological evolution. Compared to the traditional optimization methods, such as calculus-based and enumerative stragegies, EAs are robust, global and can be generally applied without recourse to domain-specific heuristics. But the characteristics of population diversity and selective pressure are not easy to be implemented in EAs such as evolutionary programming(EP), evolution strategies(ES) and genetic algorithm(GA), simultaneously. As selective pressure is increased, the search focuses on the top individuals in the population, but because of this exploitation genetic diversity is lost. The reason is that the representations of EAs are defined using deterministic values.

The new evolutionary algorithm which uses stochastic representation was proposed in [1]. QGA are characterized by rapid convergence and global search capability, simultaneously. QGA is based on the concept and principles of quantum computing such as qubits and a linear superposition of states. One individual of QGA can represent many states at the same time, and there are weak relationships between individuals since each individual is determined by current best solution and its probability, that is, the history of individual, up to date. Because of this reason, QGA is suitable for parallel structure.

Recently, parallel evolutionary algorithms (PEAs) have been used to slove more difficult problems which need a bigger population. This results in higher computational cost. The basic motivation behind many early studies of PEAs was to reduce the processing time needed to reach an acceptable solution [2]. This was accomplished by implementing EAs on different parallel architecture. In addition, it was noted that in some cases the PEAs found better solutions than comparably sized serial EAs. There are several approaches to parallelize the serial EAs [2, 3, 4, 5].

This paper offers a new parallel evolutionary algorithm called parallel quantum-inspired genetic algorithm (PQGA). Especially, a coarse-grained parallel scheme was applied to PQGA. PQGA can reduce the computational time as compared with QGA.

This paper is organized as follows. Section 2 and 3 describe the new evolutionary algorithm, QGA and the parallel quantum-inspired genetic algorithm, respectively. Section 4 contains a description of the experiment with GAs, QGAs and PQGAs for knapsack problems for comparison purpose. Section 5 summarizes and analyzes the experimental results. Concluding remarks follow in Section 6.

2 Quantum-inspired Genetic Algorithm (QGA)

QGA 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 [6]. 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 contain information of 2^m states. However, in the act of observing a

0-7803-6657-3/01/$10.00 ©2001 IEEE

quantum state, it collapses to a single state [7].

2.1 Representation

It is possible to use a number of different representations to encode the solutions onto chromosomes in evolutionary algorithm. The classical representations can be broadly classified as: binary, numeric, and symbolic [8]. QGA 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 a superposition of states. If there is, for instance, a three-qubits system with three pairs of amplitudes such as

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

the state of the system can be represented as

$$\begin{aligned} &\frac{1}{4}|000\rangle + \frac{\sqrt{3}}{4}|001\rangle - \frac{1}{4}|010\rangle - \frac{\sqrt{3}}{4}|011\rangle \\ &+\frac{1}{4}|100\rangle + \frac{\sqrt{3}}{4}|101\rangle - \frac{1}{4}|110\rangle - \frac{\sqrt{3}}{4}|111\rangle \end{aligned} \tag{5}$$

The above result means that the probabilities to represent the state $|000\rangle$, $|001\rangle$, $|010\rangle$, $|011\rangle$, $|100\rangle$, $|101\rangle$, $|110\rangle$, and $|111\rangle$ are $\frac{1}{16}$, $\frac{3}{16}$, $\frac{1}{16}$, $\frac{3}{16}$, $\frac{1}{16}$, $\frac{3}{16}$, $\frac{1}{16}$, and $\frac{3}{16}$, respectively. By consequence, the three-qubits system of (4) contains information of eight states.

Evolutionary computing with 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 eight states, but in classical representation at least eight chromosomes, (000), (001), (010), (011), (100), (101), (110), and (111) 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 QGA

The structure of QGA is described in the following.

procedure QGA
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

QGA is a probabilistic algorithm which is similar to genetic algorithm. QGA 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 $(x_1 x_2 \cdots x_m)$. 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 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

[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.[6].

formed by using the binary solutions $P(t)$ and the best stored solution. The appropriate quantum gates can be designed in compliance with practical problems. Rotation gates such as

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

where θ is a rotation angle, will be used for knapsack problems in the next section. 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 best stored solution, the stored solution is replaced by this best solution. The binary solutions $P(t)$ are discarded at the end of the loop.

In QGA, the population size, i.e., the number of qubit chromosomes is always kept constant. This is due to conservation of qubits based on quantum computing. QGA with the qubit representation can have better convergence with diversity than conventional GAs which have fixed 0 and 1 information.

3 Parallel Quantum-inspired Genetic Algorithm (PQGA)

To find more optimized solution in shorter computing time, parallelization of EAs is indispensable. The basic motivation behind many early studies of PEAs was to reduce the processing time needed to reach an acceptable solution. This was accomplished by implementing EAs on different parallel architectures. Using many processes shortens the computing time. In the experiments reported in [3], Using n processers shortens the computation time by a factor whose value is limited to n. In the shifting balancing theory proposed by Sewall Wright [3], the evolution speed of small subpopulations with loose connection is faster than that of large population. In addition, it was noted that in some cases the PEAs found better solutions than comparably sized serial EAs, because the migration of individuals between subpopulations introduces the possibility of global search.

The parallelization of EAs has been implemented in several ways, like globally PEAs, coarse-grained PEAs and fine-grained PEAs. These are classification according to the connection topology of multi-processors [2, 3]. Also, the hybrid structure can be applied for parallelization [9]. The coarse-grained parallel scheme is considered here.

The important characteristics of coarse-grained algorithms are the use of few relatively large demes and the introduction of a migration operator. The whole population is divided into some number of populations, each subpopulation evolves independently, and the exchange of individuals, called migration, occurs after some generations. This scheme can be easily implemented and even if there is no parallel computer available, it is easy to simulate one with a network of workstation or even in a single processor machine. Utilizing migration and using some number of demes prevent converging to local optima. The performance of CGPEAs mostly depends on the migration rate, migration period and the selection of migrated individuals. The methods of migration scheme are random migration, improvement insertion, worst deletion, best migration and crossover migration.

One individual of QGA can represent many states at the same time, and there are weak relationships during individuals since each individual is determined by current best solution and its probability. Because of this characteristic, QGA is better than EAs such as EP, GA and ES for implementation of a parallel architecture. The best migration scheme was considered for the method of migration in this paper.

4 Experiment

The knapsack problem, a kind of combinatorial optimization problem, is used to investigate the performance of PQGA. 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 used in the 0-1 knapsack problem are described, and this is followed by the detailed algorithms of QGA and PQGA for the knapsack problem.

4.1 Conventional GA methods

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

In all the algorithms based on penalty functions, a binary string of the length m represents a chromosome $\mathbf{x}$ for 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]. Two types of penalties are considered, such as logarithmic penalty and linear penalty:

$$Pen_1(\mathbf{x}) = \log_2 \left(1 + \rho \left(\sum_{i=1}^{m} w_i x_i - C\right)\right),$$

$$Pen_2(\mathbf{x}) = \rho\left(\sum_{i=1}^{m} w_i x_i - C\right),$$

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.

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.

4.2 QGA for the knapsack problem

The algorithm of QGA for the knapsack problem is based on the structure of QGA proposed and it contains a repair algorithm. The algorithm can be written as follows:

procedure QGA
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 length m represents a linear superposition of solutions as in (6) to the problem. 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 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 repair algorithm of QGA 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

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

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.

used as a strategy for convergence. This **update** procedure can be described as follows:

procedure update (q)
begin
  $i \leftarrow 0$
  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 using various methods with appropriate quantum gates. It depends on a given problem.

4.3 PQGA for the knapsack problem

The structure of each processor was same as the procedure of QGA as mentioned previously. The coarse-grained parallel scheme and the best migration method were used in these experiments. In QGA, the best solution migrated from other processor cannot cause rapid convergence to possibly local optimum, since QGA has a probabilistic representation including the meaning of history of evolved individual. To reduce the load of communication between each processors, new structures were used as Figure 1 and 2. Figure 1 and 2 show the connection structures for 4 processors and 16 processors, respectively.

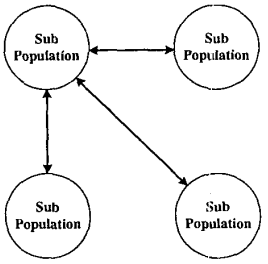

Figure 1: The connection structure for 4 processors.

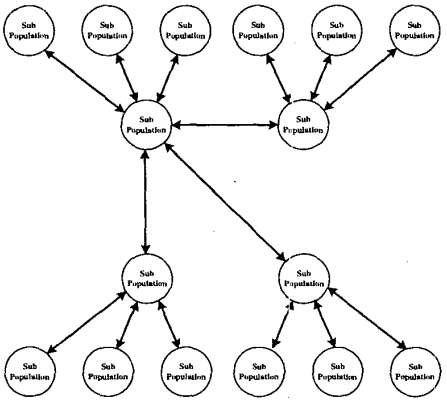

Figure 2: The connection structure for 16 processors.

5 Results

In all experiments strongly correlated sets of unsorted data were considered:

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

Average knapsack capacity given by:

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

was used. The population size considered for all the six conventional genetic algorithms (CGAs) was equal to 100. As in [10], probabilities of crossover and mutation were fixed as 0.05 and 0.01, respectively. The population size of QGA1 was equal to 10, and the population size of QGA2 was equal to 48, this being the only difference between QGA1 and QGA2. The number of processors of PQGA1 and PQGA2 were equal to 4 and 16, and the subpopulation sizes in each processor of PQGA1 and PQGA2 were equal to 12 and 3, respectively. The migration period of PQGA1 is 200 generations. In PQGA2, there are 4 sub-groups in the overall structure, and each sub-group includes 4 processors. There are 3 subpopulations in each processor. The migration between the processors occurs every 200 generations, and the

# of items			CGAs						QGAs		PQGAs	
			$Pen1$	$Pen2$	$Rep1$	$Rep2$	$Dec1$	$P2R1$	$QGA1$	$QGA2$	$PQGA1$	$PQGA2$
100	profits	b.	562.1	597.6	558.9	560.5	512.3	592.7	612.5	612.7	612.7	612.7
		m.	549.2	587.7	547.2	545.3	502.1	586.8	604.1	608.8	612.7	612.5
		w.	540.0	572.6	537.1	536.8	493.5	577.6	592.7	597.7	612.7	607.6
	$t(sec/run)$		1.537	1.544	1.293	1.302	12.97	1.556	0.408	2.354	0.596	1.112
250	profits	b.	1377.7	1455.0	1383.1	1352.7	1179.7	1454.0	1497.3	1510.2	1520.2	1515.2
		m.	1341.3	1439.0	1343.1	1337.4	1159.4	1441.7	1473.2	1495.6	1506.1	1513.3
		w.	1321.1	1415.2	1319.5	1322.5	1140.7	1430.1	1433.9	1474.6	1485.2	1494.6
	$t(sec/run)$		3.642	3.739	3.040	3.116	69.29	3.754	1.566	11.93	2.292	2.441
500	profits	b.	2712.4	2839.6	2706.8	2686.3	2255.1	2839.1	2903.2	2926.0	2965.6	3021.3
		m.	2668.4	2804.5	2661.0	2657.4	2220.9	2805.1	2858.7	2898.9	2912.3	2997.9
		w.	2642.8	2766.3	2626.6	2628.1	2195.7	2781.0	2821.1	2856.1	2882.1	2865.8
	$t(sec/run)$		9.211	9.274	7.913	8.142	270.8	9.138	4.429	34.37	7.143	6.614

Table 2: Experimental results of the knapsack problem: the maximum number of generations 1000, the number of runs 30. $P2R1$ 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.

migration between the sub-groups takes place every 500 generations. As a performance measure of the algorithm we collected the best solution found within 1000 generations over 30 runs, and we checked the elapsed time per one run. A Pentium-III $800MHz$ 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, PQGA yielded superior results as compared to all the other CGAs and QGAs. The CGA designed by using a linear penalty function and random repair algorithm outperformed all other CGAs, but is behind QGAs as well as PQGAs in performance. The results show that PQGAs and QGAs perform well in spite of small size of population. Judging from the results, PQGAs and QGAs can search solutions near the optimum within a short time as compared to CGAs, and although the total population number of PQGAs is equal to that of QGA2 (population size 48), PQGAs outperformed QGA2 in best solution and computation time. In the cases of 250 and 500 items, the experimental results again demonstrate the superiority of PQGAs.

Figure 3 shows the progress of the mean of best profits and the mean of average profits of population found by PQGA1, PQGA2, QGA1, QGA2 and CGA over 30 runs for 100, 250, and 500 items. PQGAs performs better than QGAs and CGAs in terms of convergence rate and final results. Initially, QGA2 including 48 populations shows the fastest convergence rate. PQGAs shows a slower convergence rate than QGAs due to its small population number in one processor. But in 200 generations, PQGAs outpace QGA2. It is caused by the best migration between processors. Especillay, in the cases of 250 items and 500 items, PQGA2 outperforms PQGA1 in best solution. The reason is that the structure of PQGA2 can increase the population diversity. PQGAs' final results are better than QGA's and 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, while QGA approaches towards the neighborhood of global optima with a constant convergence rate. Especially, after migrations, PQGAs have a faster convergence rate to find out global optima.

The experimental results demonstrate the effectiveness and the applicability of PQGA. Especially, figure 3 shows excellent global search ability and superiority of convergence ability of PQGA.

6 Conclusions

This paper proposed a new parallel evolutionary algorithm, PQGA with a quantum representation and a coarse-grained parallel scheme. Since QGA is based on the principles of quantum computing such as concepts of qubits and superposition of states, it can represent a linear superposition of states, and there is no need to include many individuals. QGA has 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. These characteristics of QGA are suitable for parallel structure. Also, the migration in PQGA can improve the capability of exploitation and exploration. The knapsack problem, a kind of combinatorial optimization problem, is used to discuss the performance of PQGA. It was shown that PQGA's convergence and global search ability are superior to QGA's and CGA's. The experimental results demonstrate the effectiveness and the applicability of PQGA.

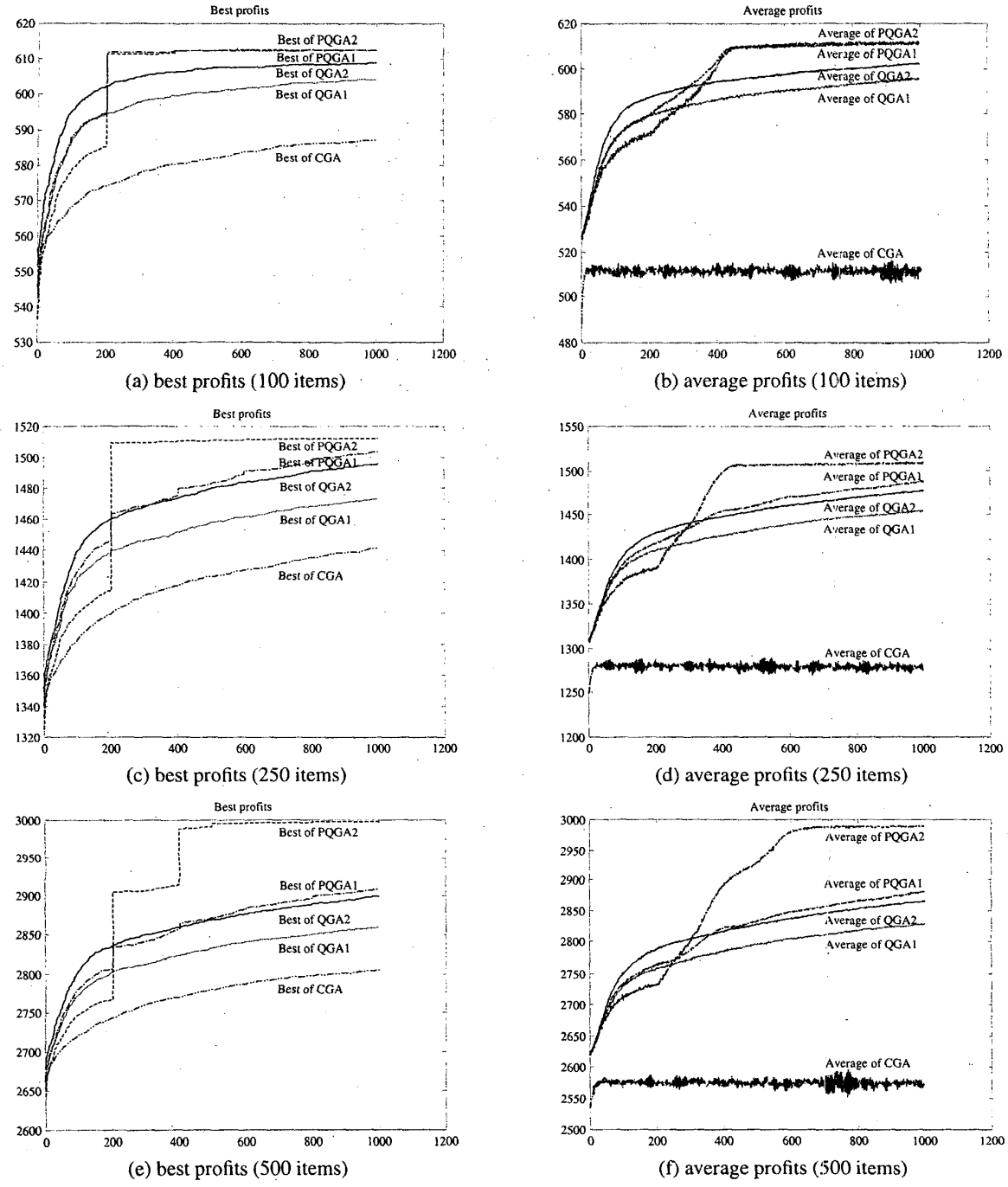

Figure 3: Comparison of PQGAs, QGAs and CGAs on the knapsack problem. The vertical axis is the profit value of knapsack, and the horizontal axis is the number of generations. (a), (c), (e) show the best profits, and (b), (d), (f) show the average profits. Both were averaged over 30 runs.

References

[1] K.-H. Han and J.-H. Kim, "Genetic Quantum Algorithm and its Application to Combinatorial Optimization Problem," in *Proceedings of the 2000 Congress on Evolutionary Computation*, pp. 1354-1360, July, 2000.

[2] E. C.-Paz, "Designing scalable multi-population parallel genetic algorithms," in *IlliGAL Report No. 98009*, Illinois Genetic Algorithms Laboratory, 1998.

[3] T. C. Belding, "The distributed genetic algorithm revisited," in *Proceedings of the Sixth International Conference on Genetic Algorithms*, pp. 114-121, Morgan Kaufmann, 1995.

[4] V. S. Gordon and D. Whitley, "Serial and parallel genetic algorithms as function optimizers," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 177-183, Morgan Kaufmann, 1993.

[5] S. Baluja, "Structure and performance of fine-grain parallelism in genetic search," in *Proceedings of the Fifth International Conference on Genetic Algorithms*, pp. 155-162, Morgan Kaufmann, 1993.

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

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

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

[9] C.-H. Lee, S.-H. Park and J.-H. Kim, "Topology and Migration Policy of Fine-grained Parallel Evolutionary Algorithms for Numerical Optimization," in *Proceedings of the 2000 Congress on Evolutionary Computation*, pp. 70-76, July, 2000.

[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.