Appl Intell
DOI 10.1007/s10489-013-0494-9

An exploratory research of elitist probability schema and its applications in evolutionary algorithms

Hong-Guang Zhang · Yuan-An Liu · Bi-Hua Tang · Kai-Ming Liu

© Springer Science+Business Media New York 2014

Abstract An important problem in the study of evolutionary algorithms is how to continuously predict promising solutions while simultaneously escaping from local optima. In this paper, we propose an elitist probability schema (EPS) for the first time, to the best of our knowledge. Our schema is an index of binary strings that expresses the similarity of an elitist population at every string position. EPS expresses the accumulative effect of fitness selection with respect to the coding similarity of the population. For each generation, EPS can quantify the coding similarity of the population objectively and quickly. One of our key innovations is that EPS can continuously predict promising solutions while simultaneously escaping from local optima in most cases. To demonstrate the abilities of the EPS, we designed an elitist probability schema genetic algorithm and an elitist probability schema compact genetic algorithm. These algorithms are estimations of distribution algorithms (EDAs). We provided a fair comparison with the persistent elitist compact genetic algorithm (PeCGA), quantum-inspired evolutionary algorithm (QEA), and particle swarm optimization (PSO) for the 0–1 knapsack problem. The proposed algorithms converged quicker than PeCGA, QEA, and PSO, especially for the large knapsack problem. Furthermore, the computation time of the proposed algorithms was less than some EDAs that are based on building explicit probability models, and was approximately the same as QEA and PSO. This is acceptable for evolutionary algorithms, and satisfactory for EDAs. The proposed algorithms are successful with respect to convergence performance and computation time, which implies that EPS is satisfactory.

H.-G. Zhang (✉) · Y.-A. Liu · B.-H. Tang · K.-M. Liu
School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing, China
e-mail: hongguang-zhang@bupt.edu.cn

Published online: 08 January 2014

Keywords Coding similarity · Elitist probability schema · Genetic algorithm · Evolutionary algorithm · Estimation of distribution algorithm

1 Introduction

To tackle complex combinatorial problems, researchers have been looking to natural principles for inspiration. During the past few decades, evolutionary algorithms (EAs) have received significant attention owing to their potential as global optimizers. Important representative algorithms in EAs include genetic algorithms (GAs) [1, 2], differential evolution [3, 4], evolutionary programming [5–7], tabu search [8–10], particle swarm optimization (PSO) [11–14], ant colony optimization [15], simulated annealing [16], quantum-inspired evolutionary algorithms (QEAs) [17–20], scatter algorithms [21, 22], bacterial foraging optimization [23], and chemical reaction optimization [24–26]. Fitness selection and random search are the basic kernels of EAs. Through iterative progress, every individual in the population adapts their coding structures to fit to the environment by a series of mechanisms such as reproduction, mutation, and recombination. Unlike traditional methods, evolutionary algorithms are fit to solve complex combinatorial problems. However, their behavior is difficult to understand or predict because of the inherent randomness. Evolutionary algorithms sometimes suffer from a lack of robustly achieving the best solution and premature convergence of the population. One of the most important problems in EAs is how to continuously find promising solutions while simultaneously escaping from local optima.

In EAs, the individuals in the population continuously come close to the optimal solution, and generally find the optimal solution or suboptimal solution before the algorithm

terminates. In the evolution process, the coding similarity of the population gradually increases. This is a basic law of EAs. The rules, principles, and rationales on the coding similarity of the population are important to the optimization of EAs and to the design of new EAs. The quantification of the coding similarity of the population is very valuable, and can be used for assessing premature convergence, guiding the search, analyzing the micro difference of multipopulations in parallel EAs, and so on.

In the following section, we will review some representative works with respect to the coding similarity of the population, which can be classified into two categories.

(a) The schema in GAs.
(b) The probability vector and probability model in the estimation of distribution algorithms (EDAs).

Holland's book [1] laid the theoretical foundation for De Jong's and all subsequent GA work by mathematically identifying the combined roles of similarity subsets, minimal operator disruption, and reproductive selection [2]. A schema is a template that identifies a subset of strings with similarities at certain string positions [1]. Schema can describe the similarities at some string positions definitely, such as 0s at 5th string position and 1s at 6th string position. However, at other string positions, the wildcard symbol $*$ is used, which means that these positions have a value of either 1 or 0. In fact, the similarity at every string position changes gradually and successively during the whole evolution process. The expression format of schema lacks flexibility and detailed information of the similarity of binary strings.

EDAs explore the solution space by building probability vectors or probability models on promising candidate solutions. Probabilistic methods in EDAs range from probability vectors to Bayesian networks (BNs). Important representative algorithms in EDAs include the univariate marginal distribution algorithm [27], the mutual information maximization for input clustering algorithm [28], mutual information trees algorithm [29, 30], bivariate marginal distribution algorithm [31], the factorized distribution algorithm [32], compact genetic algorithm (CGA) [33–37], and the Bayesian optimization algorithm (BOA) [38]. Pelikan and Goldberg summarized existing research on population-based probabilistic search algorithms that model promising solutions by estimating their probability distributions, and they used the constructed model to guide the exploration of the search space [39].

We will use CGAs and BOA to demonstrate the difference between schema and probability vectors or BNs.

Inspired by the random walk model introduced by Harik [33], a CGA was proposed in [34]. CGA represents the population as a probability vector. Let two individuals compete according to their fitness and the preferred schema. Then, update the probability vector to the better individual, and generate new individuals according to the new probability vector. CGA finds better solutions by updating the probability vector. In essence, the probability vector expresses the similarity of binary strings. The probability vector can quantify the coding similarity at each string position. However, the updating step length of the probability vector in CGA is equal to $1/(\textit{population size})$, which is a constant. To some extent, this has effect on the accuracy of the coding similarity of the population. Harik also proposed an extended CGA [35] that combined a greedy marginal product model search algorithm with a minimal description length search model. Ahn and Ramakrishna proposed elitism-based CGAs [36], which include the persistent elitist compact genetic algorithm (PeCGA) and the nonpersistent elitist CGA. Lee proposed CGAs on belief vectors that use belief vectors instead of probability vectors [37]. Each element of the belief vectors has a probability distribution with a mean and a variance.

The Bayesian optimization algorithm uses BNs to capture the (in)dependencies among the decision variables of the optimization problem [38]. The main methods of BOA are building a BN that models promising solutions, and generating new solutions by sampling from the probability distribution encoded by the built BN. The BN is an acyclic directed graph with one node per variable which is capable of capturing more complex problem decompositions and providing additional information about the coding similarity of the population. However, building explicit probability models in EDAs is often more time consuming.

Based on the above remark, we propose an elitist probability schema (EPS) for the first time, to the best of our knowledge. An EPS is an index of the coding similarity of binary strings, which can express the similarity of an elitist population at every string position. One of our key innovations with respect to existing work is that EPS can continuously predict promising solutions while simultaneously escaping from local optima in most cases. EPS is more flexible and accurate than the existing schema and probability vector methods. Furthermore, EPS is easy to use and cheap to compute because it does not rely on a complicated probability model.

The focus of this paper is to research the rationale of the EPS with respect to the coding similarity of populations, and to demonstrate the use of the EPS in EAs. The main research of this paper can be summarized as: (a) the rationale of the EPS; (b) the use of the EPS in GAs; and (c) the use of the EPS in CGA. We have designed an elitist probability schema genetic algorithm (EpsGA) and an elitist probability schema compact genetic algorithm (EpsCGA) to demonstrate the applications of the EPS. We provide a fair comparison with PeCGA, QEA, and PSO in the 0–1 knapsack problem. Our results show that EpsGA and EpsCGA significantly outperform the other algorithms.

Individual No.1=	0	0	1	1
Individual No.2=	0	0	1	1
Individual No.3=	1	1	0	0
Individual No.4=	1	1	0	0
EPS=	0.50	0.50	0.50	0.50
	0.50	0.50	0.50	0.50

(a) the first generation

Individual No.1=	0	0	1	0
Individual No.2=	1	0	0	1
Individual No.3=	1	0	1	0
Individual No.4=	1	1	0	1
EPS=	0.25	0.75	0.50	0.50
	0.75	0.25	0.50	0.50

(b) the middle generation

Individual No.1=	1	0	0	0
Individual No.2=	1	0	0	1
Individual No.3=	1	0	0	1
Individual No.4=	1	0	1	1
EPS=	0	1.00	0.75	0.25
	1.00	0	0.25	0.75

(c) the last generation

Fig. 1 A simple example of elitist probability schema during the evolution process

The rest of the paper is organized as follows. In Sect. 2, we introduce EPS and its applications. In Sects. 3 and 4, we choose the 0–1 knapsack problem as the test problem and choose PeCGA, QEA, and PSO as compared algorithms. In Sects. 5 and 6, experiment results are given. Section 7, we present concluding remarks.

2 Elitist probability schema and its applications

2.1 Elitist probability schema

An EPS, P_s, is an index of binary strings that can express the similarity of an elitist population at every string position, as shown in Eq. (1).

$$P_s = \begin{bmatrix} p_{01} & p_{02} & \cdots & p_{0i} & \cdots & p_{0N} \\ p_{11} & p_{12} & \cdots & p_{1i} & \cdots & p_{1N} \end{bmatrix} \tag{1}$$

$$p_{0i} = \frac{m_{0i}}{M} \tag{2}$$

$$p_{1i} = \frac{m_{1i}}{M} \tag{3}$$

$$\forall p_{0i}, p_{1i} \quad (1 \le i \le N)\ p_{0i} + p_{1i} = 1 \tag{4}$$

where N is the length of the binary coding. M is the number of individuals in the elitist population. m_{0i} and m_{1i} are the number of individuals with 0s at the ith string position and 1s at the ith string position, respectively ($m_{0i} + m_{1i} = M$).

We illustrate the rationales of the EPS in the evolution process using a simple example. As shown in Fig. 1, the coding similarity of the population increases gradually in EAs, and EPS can express the coding similarity of the population more flexibly and accurately than schemas and probability vectors. It is worth noticing that, for each generation, EPS can objectively quantify the coding similarity of a population.

EPS is only based on the coding of populations, and is not related to the fitness function. However, EPS expresses the accumulative effect of fitness selection with respect to the coding similarity of the population. The elitist population determines the inherent property of the EPS. A prominent feature of the EPS is that it can continuously provide the search direction of the promising solutions. In essence, each pair of p_{0i} and p_{1i} expresses a judgment of the promising solutions at the ith string position.

As the elitist population increases, the reliability and accuracy of the EPS improves. However, the computation cost also gradually increases. Generally, we suggest an elitism population of more than 20 for combinatorial problems.

2.2 Elitist probability schema genetic algorithm

EpsGA contains an elitist probability schema selection operator (EPSSO), an elitist probability schema crossover operator (EPSCO), and an elitist probability schema mutation operator (EPSMO).

2.2.1 Elitist probability schema selection operator

Each pair of p_{0i} and p_{1i} in the EPS expresses a judgment of the promising solutions at the ith string position. For example, $p_{0i} = 0.7$ and $p_{1i} = 0.3$ represent 70 % agreement with 0s at the ith string position and 30 % agreement with 1s at the ith string position. Therefore, we compute the weighted fitness of each individual using the EPS to improve the efficiency of fitness selection. In EPSSO, the fitness of each individual is weighted as shown in Eq. (5).

$$f_w = f \times \frac{\sum_{i=1}^{P} w_{ci}}{\sum_{i=1}^{P} w_{ei}} \tag{5}$$

$$E_I = \{e_1 \quad e_2 \quad \ldots \quad e_i \quad \ldots \quad e_N\} \tag{6}$$

$$e_i = \begin{cases} 0 & p_{1i} < p_{0i} \\ 1 & p_{0i} \le p_{1i} \end{cases} \tag{7}$$

$$w_{ei} = \frac{\max(p_{0i}, p_{1i})}{\min(p_{0i}, p_{1i})} \tag{8}$$

Fig. 2 The procedure of elitist probability schema genetic algorithm

Parameters: *population size, coding length, replacement rate* p_r, *crossover probability* p_c, *mutation probability* p_m
Step 1: *Initialize an initial population and evaluate the fitness.*
Step 2: *According to the replacement rate* p_r, *obtain elitist population.*
Step 3: *Compute elitist probability schema.*
Step 4: *Perform* ***EPSSO***.
Step 5: *Breed the new offsprings through* ***EPSCO*** *and* ***EPSMO***.
Step 6: *Evaluate the fitness of the new population.*
Step 7: *Check whether the stop condition is satisfied or not. If it is not satisfied, go to* ***Step 2***.
Step 8: *The individual with the best fitness represents the final solution.*

$$w_{ci} = \begin{cases} \frac{\max(p_{0i}, p_{1i})}{\min(p_{0i}, p_{1i})} & c_i = e_i \\ \frac{\min(p_{0i}, p_{1i})}{\max(p_{0i}, p_{1i})} & c_i \neq e_i \end{cases} \tag{9}$$

where f_w is the weighted fitness, f is the fitness, and P is the population size. We define E_I as the elitist individual based on the EPS. For an individual, c_i is the value at the ith string position of the binary string. w_{ei} and w_{ci} are the weighting factors for e_i and c_i, respectively.

EPSSO is based on a roulette wheel selection, and its procedure is as follows.

Step 1: Compute the weighted fitness of each individual according to Eq. (5).
Step 2: Perform a roulette wheel selection according to the weighted fitness.

2.2.2 Elitist probability schema crossover operator

The crossover operator is an important operator in GAs, and it has a profound effect on the quality of new offspring. EPSCO is based on a uniform random crossover mechanism. This mechanism enhances the randomness of the search and enlarges the search region of promising new offspring.

Based on the EPS, generate a temporary offspring, O_e, using

$$O_e = \{o_{e1} \quad o_{e2} \quad \ldots \quad o_{ei} \quad \ldots \quad o_{eN}\} \tag{10}$$

$$o_{ei} = \begin{cases} 0 & 0 \leq r_i \leq p_{0i} \\ 1 & p_{0i} < r_i \leq 1 \end{cases} \tag{11}$$

where r_i is a random number uniformly distributed in (0, 1).

The procedure used by EPSCO is

$$P_a = \{p_{a1} \quad p_{a2} \quad \ldots \quad p_{ai} \quad \ldots \quad p_{aN}\} \tag{12}$$

$$O_o = \{o_{o1} \quad o_{o2} \quad \ldots \quad o_{oi} \quad \ldots \quad o_{oN}\} \tag{13}$$

$$o_{oi} = \begin{cases} o_{ei} & 0 \leq r_i \leq 0.5 \\ p_{ai} & 0.5 < r_i \leq 1 \end{cases} \tag{14}$$

where P_a is a parent's string, O_o is a new offspring based on P_a and O_e, and r_i is a random number uniformly distributed in (0, 1).

2.2.3 Elitist probability schema mutation operator

EPSMO ensures that the mutation of the chosen genome is based on the EPS to avoid premature convergence of the population. The chosen genome, c_i, corresponds to the value at the chosen string position. In EPSMO, the mutation process of c_i is given by

$$c_i = \begin{cases} 1 & p_{0i} > p_{1i} \\ 0 & p_{0i} \leq p_{1i} \end{cases} \tag{15}$$

2.2.4 Algorithm procedure

The EpsGA procedure is given in Fig. 2.

2.3 Elitist probability schema compact genetic algorithm

The EpsCGA is based on the CGA framework. EpsCGA updates the EPS to search the solution space.

2.3.1 Compact genetic algorithm

CGA searches by updating a probability vector, which enables it to use less memory than other algorithms. The CGA procedure is shown in Fig. 3.

2.3.2 Modification

We have modified the existing algorithm to use the EPS.

In CGA, the updating step length of the probability vector is $1/(\textit{population size})$, which is a constant. However, the updating step length of the EPS in EpsCGA is variable. This is useful as it improves the algorithm's flexibility when searching the solution space.

There are only two individuals in one generation in CGA, so it is not satisfactorily robust. To improve the robustness of the proposed algorithm, the search process in EpsCGA uses a multi-individual population. The elitist individual E_I (based on the EPS) and a temporary elitist individual TE_I (based on the temporary elitist probability schema

Fig. 3 The procedure of compact genetic algorithm

Parameters: *population size, coding length*
Step 1: *Initialize probability vector*
for $i := 1$ *to coding length do* $p[i] := 0.5$;
Step 2: *Generate two individuals from the probability vector*
$a :=$ ***generate***(p); $b :=$ ***generate***(p);
Step 3: *Let them compete*
winner, loser := ***compete***(a, b);
Step 4: *Update the probability vector*
for $i := 1$ *to coding length do*
if $winner[i] \neq loser[i]$ *then*
if $winner[i] == 1$ *then* $p[i] := p[i] + 1/$*population size;*
else $p[i] := p[i] - 1/$*population size*;
Step 5: *Check whether the stop condition is satisfied or not. If it is not satisfied, go to* ***Step 2***.
Step 6: *The probability vector represent the final solution.*

Parameters: *population size, coding length, elitist probability* p_e,
elitist probability schema limit l_{EPS}
Step 1: *Initialize elitist probability schema* ***EPS*** *and the elitist individual* $\boldsymbol{E_I}$.
Step 1-1: *Initialize an initial population and evaluate the fitness.*
Step 1-2: *According to elitist probability* p_e, *obtain elitist population.*
Step 1-3: *Compute elitist probability schema EPS.*
EPS := ***compute***(*elitist population*)
Step 1-4: *Initialize the elitist individual* E_I *based on EPS as shown in Eqs. (6)–(7).*
$\boldsymbol{E_I}$:= ***initialize***(*EPS*)
Step 2: *Compute a temporary elitist probability schema* ***TEPS*** *and a temporary elitist individual* $\boldsymbol{TE_I}$
Step 2-1: *Generate a temporary population based on EPS and evaluate the fitness.*
a temporary population := ***generate***(*EPS*)
Step 2-2: *According to elitist probability* p_e, *obtain a temporary elitist population.*
Step 2-3: *Compute a temporary elitist probability schema TEPS.*
TEPS := ***compute***(*a temporary elitist population*)
Step 2-4: *Compute* TE_I *based on TEPS as shown in Eqs. (6)–(7).*
$\boldsymbol{TE_I}$:= ***compute***(*TEPS*)
Step 3: *Let* $\boldsymbol{E_I}$ *and* $\boldsymbol{TE_I}$ *compete according to the fitness and choose winner and loser.*
If E_I *is better than* TE_I
winner := *EPS*; *loser* := *TEPS*;
else
winner := *TEPS*; *loser* := *EPS*;
Step 4: *Update EPS*
EPS := ***Update***(*winner, loser,* l_{EPS}, *coding length*);
Step 5: *Check whether the stop condition is satisfied or not. If it is not satisfied, go to* ***Step 2***.
Step 6: *The elitist individual based on winner represents the final solution.*

Fig. 4 The procedure of elitist probability schema compact genetic algorithm

(TEPS)) compete with each other according to their fitness.

CGA checks whether the probability vector has converged or not. We do not check the convergence of the EPS because we use the elitist probability schema limit, l_{EPS}, as a limit of the EPS. This limit can prevent EpsCGA from falling into a local optimum, especially in early generations.

2.3.3 Algorithm procedure

The EpsCGA procedure is given in Fig. 4. An important kernel of EpsCGA is the update of the EPS, and its procedure is given in Fig. 5.

In essence, the winner and loser express their judgment of the promising solutions at each string position. Important references for updating EPS are the winner direction *WD*,

Fig. 5 The procedure of updating the elitist probability schema

```
Parameters: winner, loser, elitist probability schema limit l_EPS, coding length
Step 1: Compute the winner direction WD and the loser direction LD.
        for i := 1 to coding length do WD[i] := winner[0, i] > winner[1, i];
        for i := 1 to coding length do LD[i] := loser[0, i] > loser[1, i];
Step 2: Compute EPS after updating
        for i := 1 to coding length do
            /*the search directions are the most important reference to update EPS*/
            if WD[i] == LD[i]                    // The first IF conditional branch
                /*update the EPS steadily*/
                EPS[0, i] = (winner[0, i] + loser[0, i])/2;
                EPS[1, i] = (winner[1, i] + loser[1, i])/2;
                if EPS[0, i] > l_EPS {EPS[0, i] = l_EPS; EPS[1, i] = 1 − l_EPS; }
                if EPS[1, i] > l_EPS {EPS[1, i] = l_EPS; EPS[0, i] = 1 − l_EPS; }
            else                                 // The first ELSE conditional branch
                /*if the winner direction is not same to the loser direction, it
                demonstrates the winner is not good enough to guide the search*/
                if max(winner[0, i], winner[1, i]) > max(loser[0, i], loser[1, i])
                    EPS[0, i] = loser[0, i]; EPS[1, i] = loser[1, i];
                else
                    EPS[0, i] = winner[0, i]; EPS[1, i] = winner[1, i];
```

and the loser direction *LD* (Fig. 5). The EPS update is as follows.

(a) Generally, *WD* and *LD* are the same in the evolution process, because the winner is based on the elitist population, and the loser is based on a temporary elitist population. Therefore, if *WD* is the same as *LD*, we should steadily update the EPS as shown in Fig. 5 (The first IF conditional branch).
(b) The difference between *WD* and *LD* is important. It demonstrates that the winner is not good enough to guide the search at some string positions. Therefore, we should carefully update the EPS at these string positions, as shown in Fig. 5 (The first ELSE conditional branch).

3 Test problem

3.1 Knapsack problem

The knapsack problem is a classical NP-hard problem in combinatorial optimization that can model many industrial situations such as cargo loading [40, 41]. We have chosen the 0–1 knapsack problem as our test problem. Suppose we are given the following parameters: p_i is the profit of each type-i item ($1 \leq \mathrm{i} \leq N$), w_i is the weight of each type-i item ($1 \leq \mathrm{i} \leq N$), and C is the weight capacity of the knapsack.

Then, we can formulate the 0–1 knapsack problem as

$$\text{Maximize} \sum_{i=1}^{N} p_i x_i \tag{16}$$

subject to:

$$\sum_{i=1}^{N} w_i x_i \leq C \tag{17}$$

where $x = \{x_1 x_2 \ldots x_i \ldots x_N\}$ is a binary decision variable that defines a solution. If a type-i item is loaded in the knapsack, $x_i = 1$, otherwise, $x_i = 0$.

3.2 Test dataset

The OR-Library [42] is an important collection of test datasets for a variety of operational research problems. The OR-Library includes the classical datasets of the multidimensional knapsack problem and classical datasets of the multidemand multidimensional knapsack problem. With respect to the 0–1 knapsack problem, there are also some classical datasets and classical methods of generating test datasets as follows.

(a) The test datasets of Kreher and Martello that are widely regarded as classical datasets of the 0–1 knapsack problem [43–45];
(b) Classical methods of generating test datasets for the 0–1 knapsack problem that were proposed by Martello, Toth, and Pisinger [46–48].

Except for the test datasets of Kreher and Martello [43–45], we have used the uncorrelated, weakly correlated, and strongly correlated methods for generating test datasets [46–48]. These methods have been frequently used by other researchers [17, 18, 26, 49].

Table 1 Summary of different test datasets with respect to knapsack problems

Source	Contributor	Datasets and Methods of generating test datasets
OR-Library [42]	Cappanera	Datasets of multidemand multidimensional knapsack problem
	Petersen	Datasets of multidimensional knapsack problem
Florida State University [43]	Martello Toth	Datasets of the 0–1 knapsack problem (p01–p08)
University of Copenhagen [46]	Martello Pisinger Toth	Methods of generating the 0–1 knapsack problem datasets (uncorrelated, weakly correlated, strongly correlated, inverse strongly correlated, almost strongly correlated, subset-sum, even–odd subset-sum, even–odd strongly correlated, uncorrelated with similar weights, avis subset-sum, avis knapsack, collapsing knapsack, bounded strongly correlated, no small weight)

(a) Uncorrelated: There is no correlation between the profit and weight of each item. The profit p_i, and the weight w_i are randomly distributed in $(10, R)$.
(b) Weakly correlated: The weight w_i is randomly distributed in $(1, R)$, and the profit p_i $(p_i \geq 1)$ is randomly distributed in $(w_i - R/10, w_i + R/10)$.
(c) Strongly correlated: Such datasets correspond to real-life situations where the return is a linear function of the investment plus (or minus) some fixed charge incurred by each project. The weight w_i is randomly distributed in $(1, R)$, and the profit p_i is set to $w_i + 10$.

In this paper, we use $R = 100$ and $N = 1000$, 10000, and 20000. The weight capacity of the knapsack C is set to half the total weight.

$$C = 0.5 \sum_{i=1}^{N} w_i \tag{18}$$

3.3 Constraint handling

The constraint handling procedures have a significant impact on experiment results of the 0–1 knapsack problem. Our objective was to objectively compare the convergence of different algorithms. Therefore, we used the simple repair method from [50].

If any capacity constraint was violated, we removed items one by one from a solution x of the 0–1 knapsack problem. The order of deletion was determined by the profit/weight ratio of each item, as shown in Eq. (19). The items that had the lowest profit/weight ratio were removed first. This mechanism satisfies the capacity constraints and diminishes the overall profit as little as possible.

$$q_i = \frac{p_i}{w_i} \quad 1 \leq i \leq N \tag{19}$$

3.4 Summary of test problem

We have chosen the 0–1 knapsack problem as our test problem. The test dataset and constraint handling method are important factors when solving the 0–1 knapsack problem. A good constraint handling method can significantly improve the convergence performance of the proposed algorithm. However, to objectively compare the convergence performance of different algorithms, we choose the simple repair method from [50]. Table 1 contains a summary of different test datasets.

4 Compared algorithms

We choose PeCGA [36], QEA [18], and PSO [12] as reference algorithms to give comprehensive comparisons.

4.1 Persistent elitism compact genetic algorithm

PeCGA uses the persistent elitism mechanism instead of the random generation mechanism of CGA. The PeCGA procedure is shown in Fig. 6.

4.2 Quantum-inspired evolutionary algorithm

QEA is based on concepts and principles of quantum computing such as qubits and superposition of states. The quantum gates $U(\theta)$ in QEA are shown in Eq. (20), where θ is $s(\alpha_i \beta_i) * Delta$ in Table 2.

$$U(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{20}$$

4.3 Particle swarm optimization

PSO is based on the simulation of social behaviors such as bird flocking and fish schooling. In PSO, the particle's velocity vector is updated using

$$V_t = V_{t-1} + c_1{}^* f_1.{}^*(L_{t-1} - Y_{t-1}) + c_2{}^* f_2.{}^*(G_{t-1} - Y_{t-1}) \tag{21}$$

where V_{t-1}, Y_{t-1}, L_{t-1}, and G_{t-1} store the particle's velocity vector, position vector, personal best vector, and the

Parameters: *population size, coding length*
Step 1: *Initialize probability vector*
for $i := 1$ *to coding length do* $p[i] := 0.5$;
Step 2: *Generate one individual from the probability vector.*
if the first generation then $E_{ind} :=$ ***generate***(p); /*initialize the elite individual*/
$N_{ind} :=$ ***generate***(p); /*generate a new individual*/
Step 3: *Let them compete and let the winner inherit persistently.*
$winner, loser :=$ ***compete***(E_{ind}, N_{ind});
$E_{ind} := winner$; /*update the elite individual*/
Step 4: *Update the probability vector*
for $i := 1$ *to coding length do*
if $winner[i] \neq loser[i]$ *then*
if $winner[i] == 1$ *then* $p[i] := p[i] + 1/population\ size$;
else $p[i] := p[i] - 1/population\ size$;
Step 5: *Check whether the stop condition is satisfied or not. If it is not satisfied, go to* ***Step 2***.
Step 6: *The probability vector represent the final solution.*

Fig. 6 The procedure of the persistent elitism compact genetic algorithm

Table 2 Lookup table of θ

x_i	b_i	$f(x) \geq f(b)$	*Delta*	$s(\alpha_i \beta_i)$			
				$\alpha_i \beta_i > 0$	$\alpha_i \beta_i < 0$	$\alpha_i = 0$	$\beta_i = 0$
0	0	*	0	*	*	*	*
1	1	*	0	*	*	*	*
0	1	*False*	*Delta*	+1	−1	0	±1
0	1	*True*	*Delta*	−1	+1	±1	0
1	0	*False*	*Delta*	−1	+1	±1	0
1	0	*True*	*Delta*	+1	−1	0	±1

Where $f()$ is the fitness function. $s(\alpha_i \beta_i)$ is the sign of θ. b_i and x_i are the ith bits of the best solution and a binary solution x, respectively.

global best vector at the $t-1$th iteration, respectively. Each vector f_1 and f_2 has a uniform distribution in $(0, 1)$ that injects randomness. The constants c_1 and c_2 are used to balance cognitive and social influences. V_t is the particle's velocity vector at the tth iteration.

The particle's position vector is updated via the sigmoid limiting transformation as shown in Eqs. (22)–(23). A maximum velocity of V_{max} is used as a limit for further exploration of the population.

$$Sigmoid(V_t) = \frac{1}{1 + e^{-V_t}} \tag{22}$$

$$Y_t = \begin{cases} 1 & r_t \leq Sigmoid(V_t) \\ 0 & r_t > Sigmoid(V_t) \end{cases} \tag{23}$$

where Y_t is the particle's position vector at the tth iteration, and r_t is a random vector uniformly distributed in $(0, 1)$ at the tth iteration.

4.4 Summary of compared algorithms

A summary of the compared algorithms is given in Table 3.

5 Experiment results

5.1 Experiment setup

The common parameters used in PeCGA, QEA, PSO, EpsGA, and EpsCGA are as follows. For each algorithm, a solution specifies a binary decision variable x of the 0–1 knapsack problem. The default termination condition was set to 300 generations. The fitness threshold value was set to 0.99 and was used as a termination condition in Sect. 5.2.3. The constraint handing of each algorithm has been discussed in Sect. 3.3. In each run, the initial populations of QEA, PSO, and EpsGA were the same and were randomly generated. The population sizes of QEA, PSO, EpsGA, and EpsCGA were set to 100.

In PeCGA, the population size was set to 20. This was different from the others because a large population size has a direct effect on the convergence speed of the PeCGA. In essence, the population size is only related to the updating step length of the probability vector in PeCGA, which is not related to the actual population or the actual population size.

In QEA, the increment of the rotation angle *Delta* of the quantum gates decreased linearly from 0.1π at the first generation to 0.005π at the last generation.

Table 3 Summary of compared algorithms

Name	Feature	Summary
PeCGA	Modifies CGA by using the persistent elitism mechanism	(a) Only requires a small amount of calculations (b) Needs less physical memory and can run in a microprocessor
QEA	Simulates concepts and principles of quantum computing such as qubits and superposition of states	(a) Uses a single elitist individual (the best solution) (b) The expression format of the Qbit population enables us to inject randomness into each individual, and can also be used to breed promising solutions
PSO	Simulates social behavior such as bird flocking and fish schooling	(a) Uses multi elitist individuals (the best solution and personal best solutions) (b) Constants c_1 and c_2 are very important for protecting population diversity

Table 4 Condition and objective of each kind of experiment

Name	Condition	Objective
Reliability	(a) Datasets: a specified dataset of 100 runs, generated randomly (b) Termination: 300 generation	To test the reliability (or repeatability) of the proposed algorithms for a specified dataset
Adaptability	(a) Datasets: randomly generated in each run (b) Termination: 300 generations	To test the adaptability of the proposed algorithms for different test datasets
Time Complexity	(a) Datasets: randomly generated in each run (b) Termination: 300 generations	To test the computation time of the proposed algorithms under the 300 generations termination condition
	(a) Datasets: p01 dataset, p07 dataset and p08 dataset (b) Termination: fitness threshold	To test the computation time of the proposed algorithms under the fitness threshold termination condition

In PSO, the two acceleration coefficients c_1 and c_2 were equal to 2, and V_{max} was equal to 6.

In EpsGA, the crossover probability p_c and mutation probability p_m were 1.0 and 0.001, respectively. EpsGA was configured to replace 80 % ($p_r = 0.8$) of its population. The 20 % ($1 - p_r$) best individuals were used as elitist population.

In EpsCGA, the elitist probability p_e was 0.2, and the size of the elitist population was 20 ((*population size*)$^* p_e$). The elitist probability schema limit, l_{EPS}, was set to 0.9.

5.2 Experiment results

To fully demonstrate the abilities of the proposed algorithms, we performed three kinds of experiments to test reliability, adaptability, and time complexity. The conditions and objectives of each kind of experiment are given in Table 4.

5.2.1 Reliability

To evaluate the reliability of the proposed algorithms, we executed 100 runs for a specified dataset of under 1000 items ($N = 1000$), which was randomly generated using classical methods of generating test datasets (see Sect. 3.2).

The mean curves of the maximum, mean, and minimum fitness for 100 runs are shown in Figs. 7 and 8. The proposed algorithms produced the best solutions in all cases when compared with the other algorithms. It is worth noticing that the proposed algorithms were fast to converge, especially between the 1st and 50th generation. The experimental results of the PeCGA were not satisfactory. However, the meaning of the convergence curves for PeCGA is different from QEA, PSO, and the proposed algorithms. In PeCGA, there is only one individual in each generation, so not much calculation is required. However, the population size was 100 in QEA, PSO, and the proposed algorithms, which required many more calculations. On the other hand, the proposed algorithms are derived from the EPS, which cannot be based on a single individual. Furthermore, the ability of the EPS to predict promising solutions is proportional to the size of the elitist population. Therefore, the proposed algorithms must be based on a multi-individual population, because this is a necessary condition when using the EPS.

5.2.2 Adaptability

To test the adaptability of the proposed algorithms for different test datasets, we defined the success rates s_r of the maximum, mean, minimum, and standard deviation of the fitness using

$$s_r = \frac{N_s}{N_r}, \tag{24}$$

where N_r is the number of runs, N_s is the number of the successes of the proposed algorithms. When the maximum

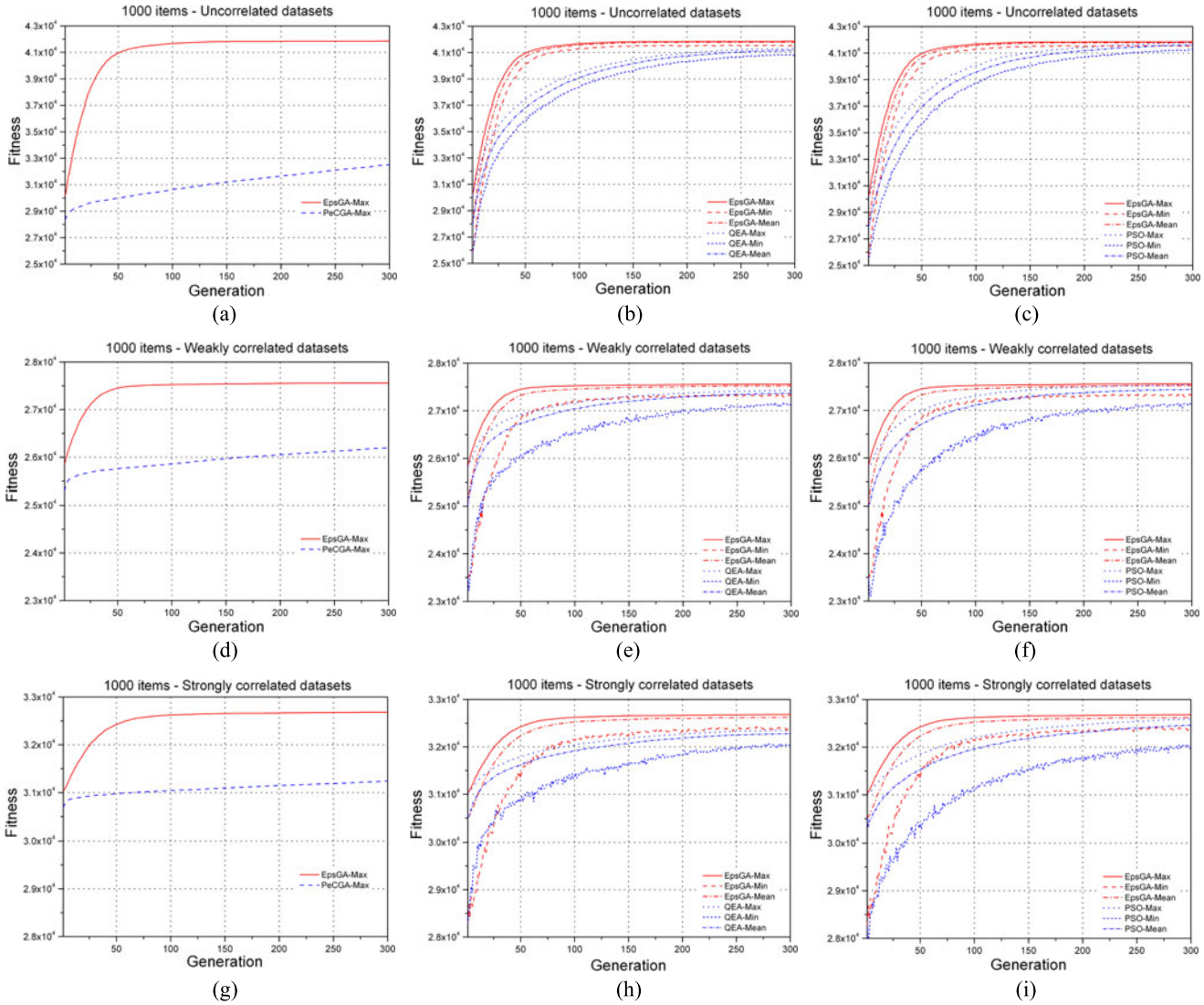

Fig. 7 Convergence curves of EpsGA

fitness of EpsGA or EpsCGA in the last generation is greater than the others in an experiment, they succeed once. The definition of N_s on the mean or minimum fitness is the same as for the maximum. When the standard deviation of the fitness in the last generation is less than the others in an experiment, EpsGA succeeds once.

We executed 100 runs ($N_r = 100$) for different test datasets of under 1000 items ($N = 1000$). As shown in Table 5, EpsGA has a success rate greater than 90 % for 23 test cases, and less than 90 % for only 4 test cases. As shown in Table 6, EpsCGA has a success rate greater than 90 % for 8 test cases, and less than 90 % for only for 1 test case. The minimum value of the success rate of EpsCGA is 80 %. These results imply that the proposed algorithms obtain consistent and satisfactory results for different test datasets.

Table 5 Success rate of EpsGA

Types		Uncorrelated datasets (%)	Weakly correlated datasets (%)	Strongly correlated datasets (%)
PeCGA	Max	100.00	100.00	100.00
QEA	Max	100.00	95.00	100.00
	Mean	100.00	100.00	100.00
	Min	100.00	100.00	100.00
	Std	100.00	93.00	72.00
PSO	Max	35.00	36.00	60.00
	Mean	93.00	97.00	100.00
	Min	91.00	93.00	98.00
	Std	97.00	98.00	95.00

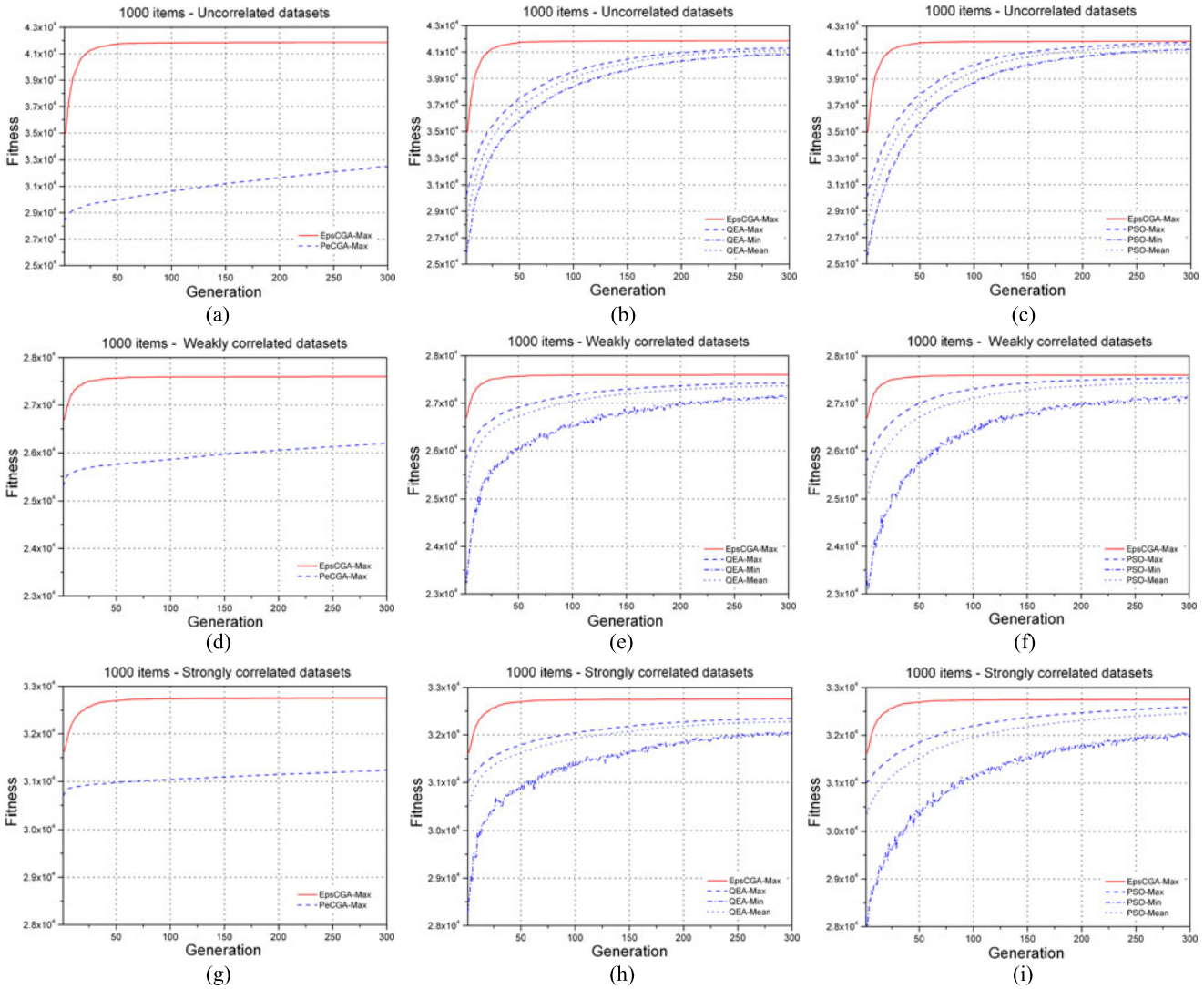

Fig. 8 Convergence curves of EpsCGA

Table 6 Success rate of EpsCGA

Types		Uncorrelated datasets (%)	Weakly correlated datasets (%)	Strongly correlated datasets (%)
PeCGA	Max	100.00	100.00	100.00
QEA	Max	100.00	98.00	100.00
PSO	Max	80.00	92.00	99.00

The statistics of the success rates are satisfactory. There are two main reasons why EPS significantly improves the classic algorithms.

(a) When we designed the EPS, it was important to validate the feasibility of basic ideas step by step under different conditions such as reference algorithms and classical test datasets. EPS is an objective index of the coding similarity of the population, as shown in Eq. (1). Furthermore, the EPS can quickly quantify the coding similarity of the population of each generation. Therefore, the definition and expression format of the EPS are very suitable for EAs.

(b) When we designed the proposed algorithms, we emphasized the importance of generating new offsprings based on the EPS such as generating a temporary offspring (O_e in Eqs. (10)–(11)) and generating a temporary population (Fig. 4). The newly generated offsprings are not only promising solutions, but they also improve the efficiency of the search of EAs for the specified region of the solution space, which is restricted by the EPS.

5.2.3 *Time complexity*

Building explicit probability models in EDAs often takes more time than in EAs. The proposed algorithms are the

Table 7 Computation time for uncorrelated datasets

Algorithm	Computation time (s)			
	Max	Mean	Min	Std
EpsGA	38.594	36.989	35.203	0.865
EpsCGA	140.687	137.431	134.843	1.424
PeCGA	0.640	0.575	0.484	0.042
QEA	96.062	93.224	90.360	1.399
PSO	104.125	102.542	101.094	1.049

Table 8 Computation time for weakly correlated datasets

Algorithm	Computation time (s)			
	Max	Mean	Min	Std
EpsGA	59.828	51.851	40.344	8.012
EpsCGA	204.328	177.675	140.500	25.226
PeCGA	0.938	0.701	0.531	0.140
QEA	146.547	122.960	91.094	19.982
PSO	164.203	138.489	114.828	20.780

Table 9 Computation time for strongly correlated datasets

Algorithm	Computation time (s)			
	Max	Mean	Min	Std
EpsGA	54.062	44.235	32.703	6.976
EpsCGA	172.547	147.400	103.000	22.689
PeCGA	0.844	0.700	0.469	0.115
QEA	139.469	119.728	66.500	20.927
PSO	146.375	123.697	99.078	17.473

union of EAs and EDAs. Therefore, it is important to demonstrate the time complexity of the proposed algorithms, which determines their practicality. Time complexity of EAs depends on its operators. Ignoring fitness functions and constraint handling methods, the proposed algorithms have a complexity of $O(G \times P \times n)$, where G is the maximum generation and P is the population size. EAs or EDAs with the same time complexity often take different computation times. Therefore, we also measured the computation times of the proposed algorithms. We implemented all algorithms in C++, and executed them on an Intel Core i7 3960X CPU and 4 GB DDR, running Visual Studio 2005 and Windows XP.

We recorded the computation times of experiments for uncorrelated, weakly correlated, and strongly correlated datasets of under 1000 items over 100 runs. The termination condition was the maximum generation. As shown in Tables 7, 8 and 9, PeCGA has the smallest mean computation time, followed by EpsGA, QEA, PSO, and EpsCGA. The computation time of the PeCGA was satisfactory. However, the experimental results of the PeCGA were not satisfactory, as discussed previously. The computation time of the EpsGA was about half that of QEA and PSO. Therefore, EpsGA is more suitable for real applications. The computation time of the EpsCGA is less than other EDAs that build explicit probability models, and is approximately the same as QEA and PSO. In some sense, this is satisfactory for EDAs.

To further demonstrate the computation time of the proposed algorithms, we performed other experiments of 100 runs on the classical datasets of Kreher and Martello [43–45]. The termination condition was the fitness threshold. As shown in Table 10, the first is the PeCGA, and the last is the EpsCGA according to the mean computation time. The computation time of the proposed algorithms is approximately the same as QEA and PSO. This is acceptable for EAs, and satisfactory for EDAs. Generally, the computation time of EAs or EDAs is often proportional to the quality of the final solution, especially for NP-hard problems.

Table 10 Computation time for classical datasets

Types		Computation time (s)			
		Max	Mean	Min	Std
p01	EpsGA	0.062	0.018	0.015	0.009
	EpsCGA	0.078	0.031	0.015	0.014
	PeCGA	0.031	0.016	0.015	0.002
	QEA	0.046	0.017	0.015	0.004
	PSO	0.031	0.017	0.015	0.004
p07	EpsGA	0.046	0.017	0.015	0.005
	EpsCGA	0.078	0.026	0.015	0.018
	PeCGA	0.031	0.016	0.015	0.003
	QEA	0.046	0.018	0.015	0.005
	PSO	0.031	0.019	0.015	0.006
p08	EpsGA	0.031	0.016	0.015	0.002
	EpsCGA	0.125	0.026	0.015	0.020
	PeCGA	0.281	0.016	0.015	0.030
	QEA	0.031	0.016	0.015	0.004
	PSO	0.031	0.017	0.015	0.004

6 Additional experiment results on large knapsack problem

6.1 Experiment setup

The population size of the PeCGA was set to 100 to adapt it to the large knapsack problem. The termination condition was set to 600 generations. The other parameters were the same as described in Sect. 5.1.

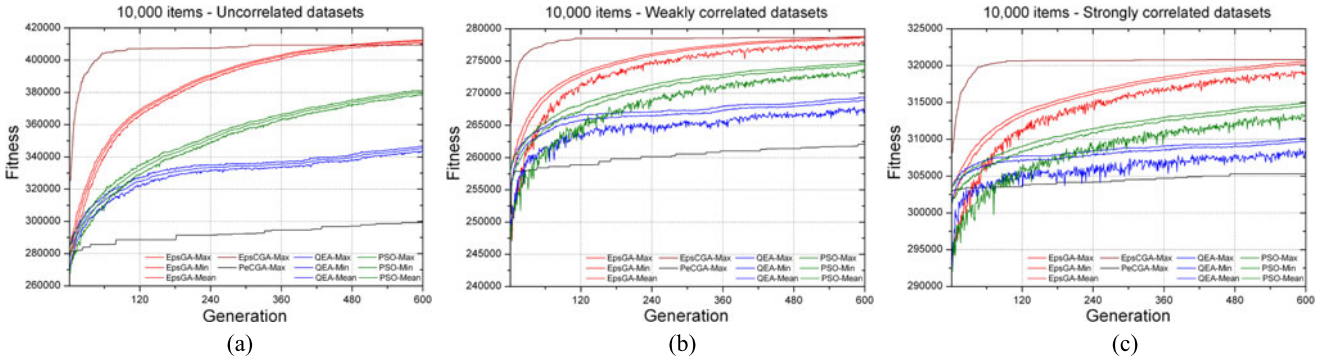

Fig. 9 Convergence curves on large knapsack problem (10000 items)

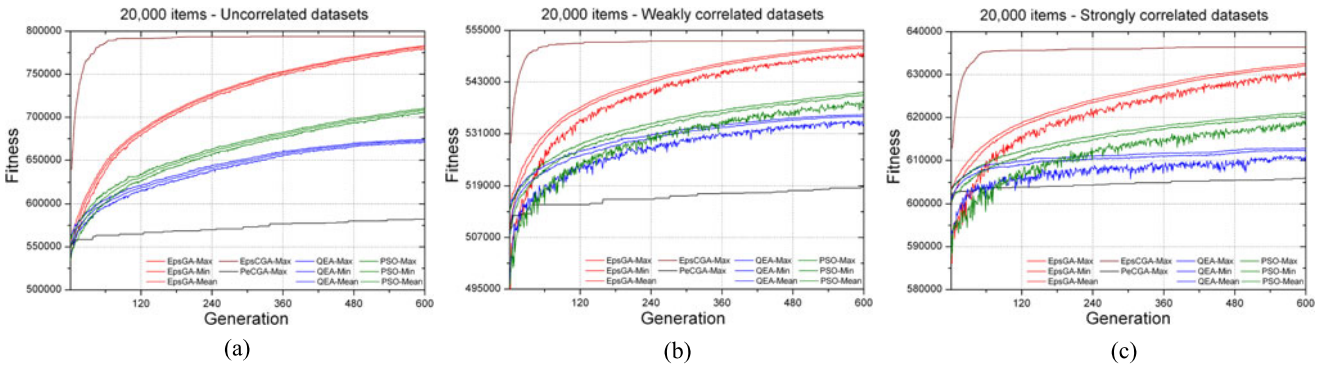

Fig. 10 Convergence curves on large knapsack problem (20000 items)

6.2 Experiment results

The large knapsack problem is one of the most difficult problems in combinatorial optimization. Generally, there are much more local optima than in the small knapsack problems. Therefore, the convergence performance of EAs on the large knapsack problem can be used to evaluate how successfully they are at escaping from local optima.

We ran experiments for the large knapsack problems (10000 items and 20000 items). The specified datasets were randomly generated using the classical methods of generating test datasets (see Sect. 3.2). The proposed algorithms significantly outperformed the other algorithms in Figs. 9 and 10. During the whole evolution process, the proposed algorithms escaped from the local optima. This is a very valuable characteristic for solving the large knapsack problems in real applications.

7 Conclusions

In EAs, it is difficult to continuously find promising solutions in the huge solution space of NP-hard problems and simultaneously protect population diversity. Fitness selection has a large effect on the coding similarity of the population. During the evolution process, the coding similarity increases gradually. This is a basic law of EAs. Therefore, the quantification of the coding similarity of the population is very valuable to EAs, because it can be used for assessing premature convergence, guiding the search, analyzing the micro difference of multipopulations in parallel EAs, and so on.

We have proposed an EPS for the first time to the best of our knowledge. An EPS is an index of binary strings, which can express the similarity of an elitist population at every string position. EPS is only based on the coding similarity of populations, and is not related to the fitness function. However, EPS expresses the accumulative effect of fitness selection with respect to the coding similarity of the population. EPS can objectively and quickly quantify the coding similarity of the population of each generation, and is more flexible and accurate than the schema in GAs and the probability vector in CGA. Furthermore, EPS is easy to use and cheap to compute because it does not rely on a complicated probability model. One of our key innovations of our methods is that EPS can continuously predict promising solutions while simultaneously escaping from local optima in most cases.

We have designed the EpsGA and EpsCGA to demonstrate the abilities of the EPS. EpsGA includes EPSSO, EPSCO, and EPSMO. EpsCGA is based on the framework of CGA, and we made modifications to use the EPS. The

0–1 knapsack problem is a classical NP-hard problem that plays important roles in many fields. Therefore, we chose the 0–1 knapsack problem as our test problem. We chose PeCGA, QEA, and PSO as reference algorithms to give comprehensive comparisons.

We have performed three kinds of experiments to test reliability, adaptability, and time complexity of the proposed algorithms. The convergence performance of the proposed algorithms is much better than PeCGA, QEA, and PSO, as shown in Fig. 7. It is worth noticing that we achieved a success rate greater than 90 % for 31 test cases, and less than 90 % for only 5 test cases (see Sect. 5.2.2). However, the computation time of EpsGA is about half that of QEA and PSO, as shown in Tables 7–9.

In addition, we have run experiments on the large knapsack problems to investigate the abilities of the proposed algorithms with respect to escaping from the local optima. The proposed algorithms significantly outperform the other algorithms, and successfully escape from the local optima in all cases, as shown in Figs. 9 and 10.

Overall, these experimental results demonstrate that EPS can continuously predict promising solutions while simultaneously escaping from local optima in most cases.

Acknowledgements The authors would like to thank the anonymous reviewers for their careful reading and constructive comments. This work was supported by National Natural Science Foundation of China (Grant No. 61272518, 61272516, 61170275), National Science and Technology Major Project of the Ministry of Science and Technology of China (Grant No. 2012ZX03001001-002), and Guangdong Provincial Science and Technology Project (Grant No. 2011B090400433).

References

1. Holland JH (1975) Adaptation in natural and artificial systems. MIT Press, Cambridge
2. Goldberg DE, Korb B, Deb K (1989) Messy genetic algorithm: motivation, analysis, and first results. Complex Syst 3:493–530
3. Kenneth P (1997) Differential evolution vs. the functions of the 2nd ICEO. In: Proc ICEC'97, pp 153–157
4. Rainer S, Kenneth P (1997) Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 4:341–359
5. Fogel LJ (1964) On the organization of intellect. Dissertation, University of California
6. Alipouri Y, Poshtan J et al (2013) A modification to classical evolutionary programming by shifting strategy parameters. Appl Intell 38:175–192
7. Liang KH, Yao X, Newton CS (2001) Adapting self-adaptive parameters in evolutionary algorithms. Appl Intell 15:171–180
8. Glover F, Taillard E, Werra DD (1993) A user's guide to tabu search. Ann Oper Res 41:1–28
9. Laguna M, Barnes JW, Glover F (1993) Intelligent scheduling with tabu search: an application to jobs with linear delay penalties and sequence-dependent setup costs and times. Appl Intell 3:159–172
10. Hedar AR, Ali AF (2012) Tabu search with multi-level neighborhood structures for high dimensional problems. Appl Intell 37:189–206
11. Kennedy J, Eberhart R (1995) Particle swarm optimization. In: Proc IEEE int conf neural netw, pp 1942–1948
12. Kennedy J, Eberhart R (1997) A discrete binary version of the particle swarm algorithm. In: Proc IEEE int conf syst man cybern, pp 4104–4108
13. Poli R, Kennedy J, Blackwell T (2007) Particle swarm optimization. Swarm Intell 1:33–57
14. Hasanzadeh M, Meybodi MR, Ebadzadeh MM (2013) Adaptive cooperative particle swarm optimizer. Appl Intell 39:397–420
15. Birattari M, Pellegrini P, Dorigo M (2007) On the invariance of ant colony optimization. IEEE Trans Evol Comput 11:732–742
16. Dekkers A, Aarts E (1991) Global optimization and simulated annealing. Math Program 50:367–393
17. Han KH, Park KH, Lee CH, Kim JH (2001) Parallel quantum-inspired genetic algorithm for combinatorial optimization problem. In: Proc ICEC'01, pp 1422–1429
18. Han KH, Kim JH (2002) Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. IEEE Trans Evol Comput 6:580–593
19. Han KH, Kim JH (2003) On setting the parameters of quantum-inspired evolutionary algorithm for practical application. In: Proc ICEC'03, pp 178–194
20. Han KH (2003) Quantum-inspired evolutionary algorithm. Dissertation, Korea Advanced Institute of Science and Technology
21. Glover F, Laguna M, Martí R (2000) Fundamentals of scatter search and path relinking. Control Cybern 29:653–684
22. Martí R, Laguna M, Glover F (2006) Principles of scatter search. Eur J Oper Res 169:359–372
23. Passino KM (2002) Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Syst Mag 22:52–67
24. Lam AYS, Li VOK (2010) Chemical-reaction-inspired metaheuristic for optimization. IEEE Trans Evol Comput 14:381–399
25. Lam AYS, Li VOK (2012) Chemical reaction optimization: a tutorial. Memetic Comp 4:3–17
26. Truong TK, Li KL, Xu YM (2013) Chemical reaction optimization with greedy strategy for the 0–1 knapsack problem. Appl Soft Comput 13:1774–1780
27. Mühlenbein H, Paass G (1996) From recombination of genes to the estimation of distributions I. Binary parameters. In: Proc int conf evol comput parallel problem solving from nature—PPSN IV, pp 178–187
28. De Bonet JS, Lsbell CL, Viola JP (1997) MIMIC: finding optima by estimating probability densities. Adv Neural Inf Process Syst 9:424–431
29. Baluja S, Davies S (1997) Using optimal dependency-trees for combinatorial optimization: learning the structure of the search space. In: Proc int conf mach learn, pp 30–38
30. Baluja S, Davies S (1998) Fast probabilistic modeling for combinatorial optimization. In: Proc 15th national conf artif intell, pp 469–476
31. Pelikan M, Mühlenbein H (1999) The bivariate marginal distribution algorithm. In: Advances in soft computing—engineering design and manufacturing, pp 521–535
32. Mühlenbein H, Mahnig T, Rodriguez A (1999) Schemata, distributions and graphical models in evolutionary optimization. J Heuristics 5:215–247
33. Harik GR, Cantú-Paz E, Goldberg DE et al (1999) The gambler's ruin problem, genetic algorithms, and the sizing of populations. Evol Comput 7:231–253
34. Harik GR, Lobo FG, Goldberg DE et al (1999) The compact genetic algorithm. IEEE Trans Evol Comput 3:287–297
35. Sastry K, Goldberg DE (2000) On extended compact genetic algorithm. IIIiGAL Report No. 2000026, Illinois Genetic Algorithms Lab
36. Ahn CW, Ramakrishna RS (2003) Elitism-based compact genetic algorithms. IEEE Trans Evol Comput 7:367–385

37. Lee JY, Kim MS, Lee JJ (2011) Compact genetic algorithms using belief vectors. Appl Soft Comput 11:3385–3401
38. Pelikan M, Goldberg DE, Cantú-paz EE (2000) Linkage problem, distribution estimation, and Bayesian networks. Evol Comput 8:311–340
39. Pelikan M, Goldberg DE, Lobo FG (2002) A survey of optimization by building and using probabilistic models. Comput Optim Appl 21:5–20
40. Kang MH, Choi HR, Kim HS, Park BJ (2012) Development of a maritime transportation planning support system for car carriers based on genetic algorithm. Appl Intell 36:585–604
41. Cho JH, Kim HS, Choi HR (2012) An intermodal transport network planning algorithm using dynamic programming—a case study: from Busan to Rotterdam in intermodal freight routing. Appl Intell 36:529–541
42. Brunel University website (2013). http://people.brunel.ac.uk/~mastjjb/jeb/info.html
43. Florida State University website (2013). http://people.sc.fsu.edu/~jburkardt/datasets/knapsack_01/knapsack_01.html
44. Kreher DL, Stinson DR (1998) Combinatorial algorithms: generation, enumeration and search. CRC Press, Boca Raton
45. Martello S, Toth P (1990) Knapsack problem: algorithms and computer implementations. Wiley, New York
46. University of Copenhagen website (2013). http://www.diku.dk/~pisinger/gen2.c
47. Martello S, Pisinger D, Toth P (1999) Dynamic programming and strong bounds for the 0–1 knapsack problem. Manag Sci 45:414–424
48. Pisinger D (1999) Core problems in knapsack algorithms. Oper Res 47:570–575
49. Kumar R, Singh PK (2010) Assessing solution quality of biobjective 0–1 knapsack problem using evolutionary and heuristic algorithms. Appl Soft Comput 10:711–718
50. Zitzler E (1999) Evolutionary algorithms for multiobjective optimization: methods and applications. Dissertation, Swiss Federal Institute of Technology Zurich

Hong-Guang Zhang received B.S., M.S., and PhD. in Electrical Engineering from Harbin Institute of Technology in 2000, 2002, and 2005 respectively. From 2006 to 2010, he was a test engineer at China Academy of Space Technology. Currently, he works at School of Electronic Engineering, Beijing University of Posts and Telecommunications, China. His research interests include artificial life, and evolutionary computation. Recently, he won the first prize award of science and technology of electronic information at Chinese Institute of Electronics.

Yuan-An Liu was a professor with School of Electronic Engineering, Beijing University of Posts and Telecommunications. His research interests include next-generation cellular system, and electromagnetic interference smart antennas for high-capacity mobile signal processing techniques. He published more than 73 SCI papers.

Bi-Hua Tang was a professor with School of Electronic Engineering, Beijing University of Posts and Telecommunications. Her research interests include wireless sensor network, and transmission technology.

Kai-Ming Liu works at School of Electronic Engineering, Beijing University of Posts and Telecommunications. His research interests include cognitive radio wireless networks, and wireless mesh networks.