Proceedings of the 2001 IEEE Congress on Evolutionary Computation Seoul, Korea • May 27-30, 2001

Scaling Up Fast Evolutionary Programming with Cooperative Coevolution

Yong Liu
The University of Aizu
Tsuruga, Ikki-machi, Aizu-Wakamatsu
Fukushima 965-8580, Japan
yliu@u-aizu.ac.jp

Xin Yao
School of Computer Science
The University of Birmingham
Edgbaston, Birmingham B15 2TT, U.K.
X.Yao@cs.bham.ac.uk

Qiangfu Zhao
The University of Aizu
Tsuruga, Ikki-machi, Aizu-Wakamatsu
Fukushima 965-8580, Japan
qf-zhao@u-aizu.ac.jp

Tetsuya Higuchi
Evolvable Systems Laboratory
Electrotechnical Laboratory
1-1-4 Umezono, Tsukuba, Ibaraki 305-8568, Japan
higuchi@etl.go.jp

Abstract- Evolutionary programming (EP) has been applied with success to many numerical and combinatorial optimization problems in recent years. However, most analytical and experimental results on EP have been obtained using low-dimensional problems. It is interesting to know whether the empirical results obtained from the low-dimensional problems still hold for high-dimensional cases. It was discovered that neither classical EP (CEP) nor fast EP (FEP) performed satisfactorily for some large-scale problems. This paper shows empirically that FEP with cooperative coevolution (FEPCC) can speed up convergence rates on the large-scale problems whose dimension ranges from 100 to 1000. Cooperative coevolution adopts the divide-and-conquer strategy. It divides the system into many modules, and evolves each module separately and cooperatively. The results of FEPCC on the problems investigated here are something of a surprise. The time used by FEPCC to find a near optimal solution appears to scale linearly; that is, the time used seems to go up linearly as the dimensionality of the problems studied increases.

1 Introduction

Evolutionary programming (EP) has been recently applied with success to many numerical and combinatorial optimization problems [1, 2]. Optimization by EP can be summarized into two major steps:

1. Mutate the solutions in the current population, and
2. Select the next generation from the mutated and the current solutions.

These two steps can be regarded as a population-based version of the classical generate-and-test method, where mutation is used to **generate** new solutions (offspring) and selection is used to **test** which of the newly generated solutions should survive to the next generation. The generate-and-test formulation of EP indicates that mutation is a key search operator which generates new solutions from the current ones.

Recently, a fast EP (FEP) algorithm based on Cauchy mutation was proposed and tested on a suite of 23 functions [3]. FEP performs much better than classical EP (CEP) for multimodal functions with many local minima while being comparable to CEP in performance for unimodal and multimodal functions with only a few local minima. However, the problems studied in [3] have 30 dimensions that are relatively small for many real-world problems. It is interesting to know whether the results obtained from the low dimensional problems can be generalized to higher-dimensional problems. It is also important to investigate the computational complexity of an EP algorithm, i.e., how it scales as the problems size increases. Unfortunately, the reported studies on the scalability of EP algorithms and evolutionary algorithms, in general, are scarce [4, 5].

It was discovered that neither classical CEP nor FEP performed satisfactorily for some large-scale problems [5]. The reason for FEP's poor performance on the high dimensional problem lies in its overly large search step size. The step size of the Cauchy mutation increases monotonically as the dimensionality increases. As pointed out in a previous study [6], there is an optimal search step size for a given problem. A step size which is too large or too small will have a negative impact on the performance of search. CEP performs poorly on the multimodal function due to its small search step size, which makes it difficult to escape from a local minimum once it is trapped.

In this paper, FEP with cooperative coevolution (FEPCC) is proposed to deal with the high dimensional problems. Cooperative coevolution adopts the divide-and-conquer strategy [7, 8]. It divides the system into many modules, and then repeats the following two steps for many generations until a good system rather than a module is obtained:

1. Evolve each module separately, and
2. Combine them to form the whole system.

0-7803-6657-3/01/$10.00 © 2001 IEEE

This paper investigates the behavior of FEPCC on large-scale problems with dimensions ranging from 100 to 1000. The problems used in our empirical studies include four unimodal functions and four multimodal functions with many local optima. The results of FEPCC on the problems investigated here are something of a surprise. FEPCC has displayed excellent robustness across dimensionality in their good performance. The time used by FEPCC to find a near optimal solution appears to scale linearly; that is, the time used seems to go up linearly as the dimensionality of the problems studied increases.

The rest of this paper is organized as follows: Section 2 describes the global minimization problem considered in this paper and FEPCC used to solve it. Section 3 presents and discusses the experimental results. Finally, Section 4 concludes with some remarks and future research directions.

2 Fast Evolutionary Programming with Cooperative Coevolution

A global minimization problem can be formalized as a pair (S, f), where $S \subseteq R^n$ is a bounded set on R^n and $f : S \mapsto R$ is an n-dimensional real-valued function. The problem is to find a point $\mathbf{x}_{min} \in S$ such that $f(\mathbf{x}_{min})$ is a global minimum on S. More specifically, it is required to find an $\mathbf{x}_{min} \in S$ such that

$$\forall \mathbf{x} \in S : f(\mathbf{x}_{min}) \leq f(\mathbf{x}),$$

where f does not need to be continuous but it must be bounded. This paper only considers unconstrained function optimization.

2.1 Function Optimization by Fast Evolutionary Programming

The FEP applied to function optimization can be described as follows [3]:

1. Generate the initial population of μ individuals, and set $k = 1$. Each individual is taken as a pair of real-valued vectors, $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, where $\mathbf{x}_i$'s are objective variables and η_i's are standard deviations for Cauchy mutations (also known as strategy parameters in self-adaptive evolutionary algorithms).

2. Evaluate the fitness score for each individual $(\mathbf{x}_i, \eta_i)$, $\forall i \in \{1, \cdots, \mu\}$, of the population based on the objective function, $f(\mathbf{x}_i)$.

3. Each parent $(\mathbf{x}_i, \eta_i)$, $i = 1, \cdots, \mu$, creates a single offspring $(\mathbf{x}_i{}', \eta_i{}')$ by: for $j = 1, \cdots, n$,

$$x_i{}'(j) = x_i(j) + \eta_i(j)\delta_j, \quad (1)$$
$$\eta_i{}'(j) = \eta_i(j) \exp(\tau' N(0,1) + \tau N_j(0,1)) \quad (2)$$

   where $\eta_i(j)$, $\eta_i{}'(j)$, $x_i(j)$, and $x_i{}'(j)$ denote the j-th component of the vectors η_i, $\eta_i{}'$, $\mathbf{x}_i$ and $\mathbf{x}_i{}'$, respectively. The factors τ and τ' are commonly set to $\left(\sqrt{2\sqrt{n}}\right)^{-1}$ and $\left(\sqrt{2n}\right)^{-1}$ [9, 10]. $N(0,1)$ denotes a normally distributed one-dimensional random number with mean 0 and standard deviation 1. $N_j(0,1)$ indicates that the random number is generated anew for each value of j. δ_j is a Cauchy random variable with the scale parameter $t = 1$, and is generated anew for each value of j. The one-dimensional Cauchy density function centered at the origin is defined by:

$$f_t(x) = \frac{1}{\pi} \frac{t}{t^2 + x^2}, \quad -\infty < x < \infty, \quad (3)$$

   where $t > 0$ is a scale parameter [11](pp.51). The shape of $f_t(x)$ resembles that of the Gaussian density function but approaches the axis so slowly that an expectation does not exist. As a result, the variance of the Cauchy distribution is infinite.

4. Calculate the fitness of each offspring $(\mathbf{x}_i{}', \eta_i{}')$, $\forall i \in \{1, \cdots, \mu\}$.

5. Conduct pairwise comparison over the union of parents $(\mathbf{x}_i, \eta_i)$ and offspring $(\mathbf{x}_i{}', \eta_i{}')$, $\forall i \in \{1, \cdots, \mu\}$. For each individual, q opponents are chosen uniformly at random from all the parents and offspring. For each comparison, if the individual's fitness is no smaller than the opponent's, it receives a "win."

6. Select the μ individuals out of $(\mathbf{x}_i, \eta_i)$ and $(\mathbf{x}_i{}', \eta_i{}')$, $\forall i \in \{1, \cdots, \mu\}$, that have the most wins to be parents of the next generation.

7. Stop if the halting criterion is satisfied; otherwise, $k = k + 1$ and go to Step 3.

The FEP described is similar to CEP in [9] except for Eq.(2) which is defined in CEP:

$$x_i{}'(j) = x_i(j) + \eta_i(j) N_j(0,1) \quad (4)$$

Figure 1 shows the difference between the Cauchy mutation and Gaussian mutation. It is clear from Figure 1 that Cauchy mutation is more likely to generate an offspring further away from its parent than Gaussian mutation due to its long flat tails. It is expected to have a higher probability of escaping from a local optimum or moving away from a plateau, especially when the "basin of attraction" of the local optimum or the plateau is large relative to the mean step size. On the other hand, the smaller hill around the center in Figure 1 indicates that Cauchy mutation spends less time in exploiting the local neighborhood and thus has a weaker fine-tuning ability than Gaussian mutation in small to mid-range regions. The empirical results support the above intuition [3] .

2.2 Scaling Up Performance by Cooperative Coevolution

Given that a solution to a function minimization problem consists of a vector of n variable values, a direct cooperative coevolution for function optimization is to assign a population

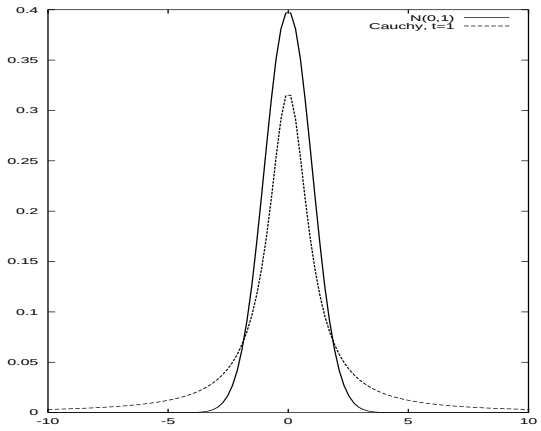


Figure 1: Comparison between Cauchy and Gaussian density functions.

to each variable, and coevolve the individuals in each separate population [7, 8]. One complete evolution of all populations for n variables during one generation is called a *cycle*. A cycle of evolution of FEPCC consists of three major steps:

1. $j = 1$. Apply FEP to the population of variable 1.
2. $j = j + 1$. Apply FEP to the population of variable j.
3. If $j \leq n$, go to Step 2. Otherwise, go to the next cycle.

To reduce the evaluation time, the fitness of an individual in FEPCC was estimated by combining it with the current best individuals from each of the other populations to form a vector of real values, and applying the vector to the target function. The fitness evaluation in FEPCC is different with that in FEP. In FEP, the target function value of a new individual have to be recalculated since an individual is a vector and the most of components in the vector have new values. In contrast, because an individual in FEPCC is a component in a vector and other components in the vector remain unchanged, it only needs to calculate the difference caused by the changed component. This fitness evaluation method used in this paper takes less computation time.

3 Experimental Studies

3.1 Benchmark Functions

Eight benchmark functions [1, 9, 12, 13] were used in our experimental studies. The eight benchmark functions are given in Table 1. Functions f_1 to f_4 are unimodal. Functions f_5 to f_8 are multimodal functions where the number of local minima increases exponentially with the problem dimension [12, 13].

3.2 Experimental Setup

In experiments, the population size $\mu = 50$, the tournament size $q = 10$ for selection, and the initial $\eta = 3.0$ were used. These parameters follow the suggestions from Bäck and Schwefel [9] and Fogel [1]. The initial population was generated uniformly at random in the range as specified in Table 1.

3.3 Unimodal Functions

The first set of experiments was aimed to study the convergence rate of FEPCC for functions f_1 to f_4 with different dimensions. The dimensions of f_1–f_4 were set to 100, 250, 500, 750, and 1000, respectively, in our experiments. For a function in each dimension, 50 independent runs were conducted. The average results of 50 runs are summarized in Table 2. Figure 2 shows the progress of the mean solutions found over 50 runs for f_1 to f_4. Function f_1 is the simple sphere model studied by many researchers. Function f_3 is a discontinuous function. FEPCC maintains a nearly constant convergence rate throughout the evolution for functions f_1 and f_3. For function f_2, when the term of $\prod_{i=1}^{n} |x_i|$ dominates the target function value, FEPCC displays a fast convergence rate. Once the term of $\sum_{i=1}^{n} |x_i|$ takes over, FEPCC's convergence rate reduces substantially. Function f_4 is the step function that is characterized by plateaus and discontinuity. FEPCC moves quickly from one plateau to a lower one for f_4.

According to Figure 2 and Table 2, the computational time used by FEPCC to find an optimal solution to the unimodal functions appears to grow in the order of O(n). For example, for function f_1, 500000 fitness evaluations were need for $n = 100$, 1250000 for $n = 250$, 2500000 for $n = 500$, 3750000 for $n = 750$, and 5000000 for $n = 1000$. Although the number of fitness evaluation is relatively large in FEPCC, the totol computation time is considerably short. An individual in a population in FEPCC is a single variable. Its fitness can be easily evaluated from its parent. Only the difference caused by one changed variable in a vector needs to be calculated.

3.4 Multimodal Functions

Multimodal functions having many local minima are often regarded as being difficult to optimize. f_5–f_8 are such functions where the number of local minima increases exponentially as the dimension of the function increases. Five different values of n have been used in our experiments, i.e., $n = 100, 250, 500, 750$, and 1000. The average results of 50 runs are summarized in Table 3. Figure 3 shows the progress of the mean solutions found over 50 runs for f_5 to f_8. According to Figure3, FEPCC was able to improve its solution steadily for a long time for functions f_6, f_7, and f_8, but fell to a local optimum quite early for function f_5. One reason for FEPCC becoming trapped in a local optimum is that it used a *greedy* fitness evaluation. The fitness of an individual is evaluated based on the vector formed by this individual and the current best individuals from each of the other populations. To get a better evaluation, we can construct many vectors, and determine the fitness of an individual by evaluating the

Table 1: The eight benchmark functions used in our experimental study, where n is the dimension of the function, f_{min} is the minimum value of the function, and $S \subseteq R^n$.

Test function	S	f_{min}
$f_1(x) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	0
$f_2(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	$[-10, 10]^n$	0
$f_3(x) = \max_i \{\lvert x_i \rvert, 1 \leq i \leq n\}$	$[-100, 100]^n$	0
$f_4(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	$[-100, 100]^n$	0
$f_5(x) = \sum_{i=1}^{n} -x_i \sin(\sqrt{\lvert x_i \rvert})$	$[-500, 500]^n$	$-418.9829n$
$f_6(x) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10)]$	$[-5.12, 5.12]^n$	0
$f_7(x) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e$	$[-32, 32]^n$	0
$f_8(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600, 600]^n$	0

Table 2: The mean solutions found for f_1–f_4, with dimension $n = 100, 250, 500, 750$, and 1000, respectively, by FEPCC. One complete evolution of all populations in one generation is called a *cycle*. The function evaluation number in one cycle equals $100n$.

Func.	n	No. of Cyc.	Mean	Std Dev
f1	100	50	6.8×10^{-9}	1.8×10^{-8}
	250	50	2.1×10^{-8}	6.2×10^{-7}
	500	50	4.9×10^{-8}	1.2×10^{-7}
	750	50	3.4×10^{-8}	1.1×10^{-8}
	1000	50	5.4×10^{-8}	2.8×10^{-8}
f2	100	100	2.1×10^{-4}	6.1×10^{-4}
	250	100	4.3×10^{-4}	7.4×10^{-4}
	500	100	1.3×10^{-3}	3.0×10^{-3}
	750	100	2.1×10^{-3}	2.6×10^{-3}
	1000	100	2.6×10^{-3}	3.2×10^{-3}
f3	100	50	3.8×10^{-5}	4.8×10^{-5}
	250	50	5.8×10^{-5}	9.1×10^{-5}
	500	50	9.0×10^{-5}	1.5×10^{-4}
	750	50	6.0×10^{-5}	3.0×10^{-5}
	1000	50	8.5×10^{-5}	7.2×10^{-5}
f4	100	50	0.0	0.0
	250	50	0.0	0.0
	500	50	0.0	0.0
	750	50	0.0	0.0
	1000	50	0.0	0.0

target function values of all vectors containing this individual [14].

Based on Figure 3 and Table 3, the computational time used by FEPCC to find a near optimal solution to the multimodal functions also appears to grow in the order of O(n). For example, for function f_8, 500000 fitness evaluations were need for $n = 100$, 1250000 for $n = 250$, 2500000 for $n = 500$, 3750000 for $n = 750$, and 5000000 for $n = 1000$.

Table 3: The mean solutions found for f_5–f_8, with dimension $n = 100, 250, 500, 750$, and 1000, respectively, by FEPCC. One complete evolution of all populations in one generation is called a *cycle*. The function evaluation number in one cycle equals $100n$.

Func.	n	No. of Cyc.	Mean	Std Dev
f5	100	50	-41867.3	52.28
	250	50	-104677.2	96.47
	500	50	-209316.4	121.3
	750	50	-313995.8	171.1
	1000	50	-418622.6	200.6
f6	100	50	0.026	0.14
	250	50	0.048	0.15
	500	50	0.143	0.28
	750	50	0.163	0.23
	1000	50	0.313	0.40
f7	100	50	1.7×10^{-4}	2.1×10^{-5}
	250	50	3.5×10^{-4}	3.0×10^{-5}
	500	50	5.7×10^{-4}	3.9×10^{-5}
	750	50	7.8×10^{-4}	4.1×10^{-5}
	1000	50	9.5×10^{-4}	3.4×10^{-5}
f8	100	50	0.047	0.065
	250	50	0.025	0.052
	500	50	0.029	0.085
	750	50	0.061	0.195
	1000	50	0.025	0.114

Table 4: Comparison between FEPCC with FEP on function f_1 (the sphere function) with $n = 100$, 200 and 300, respectively. The results of FEPCC were averaged over 50 independent runs. The results of FEP were averaged over 10 independent runs [5].

n	FEPCC		FEP	
	No. of Function Evaluations	Mean of Solutions	No. of Function Evaluations	Mean of Solutions
100	5.0×10^5	6.8×10^{-9}	7.5×10^5	4.7×10^{-3}
200	1.0×10^6	1.4×10^{-8}	1.5×10^6	9.1×10^{-2}
300	1.5×10^6	1.6×10^{-8}	3.0×10^6	0.46

Table 5: Comparison between FEPCC with FEP on function f_8 (the Ackley's function) with $n = 100$, 200 and 300, respectively. The results of FEPCC were averaged over 50 independent runs. The results of FEP were averaged over 10 independent runs [5].

n	FEPCC		FEP	
	No. of Function Evaluations	Mean of Solutions	No. of Function Evaluations	Mean of Solutions
100	5.0×10^5	1.7×10^{-4}	7.5×10^5	3.7×10^{-2}
200	1.0×10^6	3.1×10^{-4}	1.5×10^6	15.2
300	1.5×10^6	3.6×10^{-4}	3.0×10^6	20.7

3.5 Comparisons with Fast Evolutionary Programming

Tables 4 and 5, and Figures 4 and 5 summarize the average results of FEPCC in comparison with FEP on the sphere function $f1$ and the Ackley's function $f8$. Only the results of FEP on $f1$ and $f8$, with $n = 100, 200$, and 300, respectively, were reported in [5]. When the dimensionality becomes large, FEP converges very slowly in comparison with FEPCC. The reason that FEP's performance worsens as the dimensionality increases lies in its increasingly large search step sizes. FEP's search step size (driven by the search step size of the Cauchy mutation) increases as the dimensionality increases. When the search step size is larger than the optimal one, further increase in the step size can only deteriorate the search performance [6]. FEPCC is applyed to a variable of a vector rather than the whole vector, the problem of too large step size with high dimensionality does not exist. Furthermore, the robust and faster search capability of Cauchy mutation in one dimension can be fully explored by cooperative coevolution approach.

4 Conclusions

This paper proposes FEPCC, and evaluates its scalability on a number of benchmark problems. Scalability of an algorithm is an important measurement of how good and how applicable the algorithm is. Few results on the scalability of EP algorithms and evolutionary algorithms are available at present [4, 5]. The experiment results show that the cooperative coevolution approach provides an effective and efficient way to improve scalability of FEP. The time used by FEPCC to find a near optimal solution appears to increase linearly as the dimensionality increases.

One problem was found for FEPCC is that it sometimes fell into a local optimum due to its greedy fitness evaluation. One of the future improvements to FEPCC would be to apply more complex and better fitness evaluation methods [14].

Bibliography

[1] Fogel, D. B. (1991) *System Identification Through Simulated Evolution: A Machine Learning Approach to Modeling*. Needham Heights, MA 02194: Ginn Press.

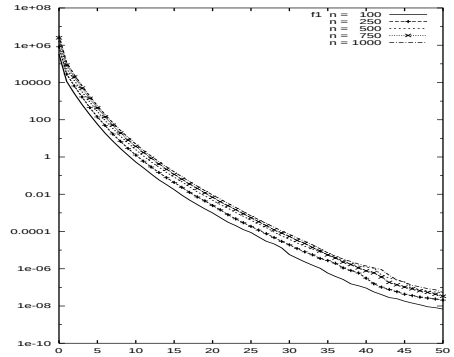
[2] Fogel, D. B. (1993) "Applying evolutionary programming to selected traveling salesman problems," *Cybernetics and Systems*, vol.24, pp.27–36.

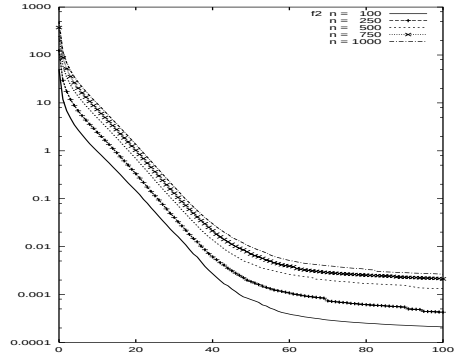
[3] Yao, X., Liu, Y., and Lin, G. (1999) "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp.82–102.

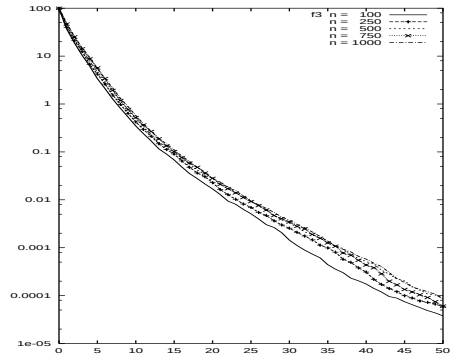
[4] Fogel, D. B. (1993) "Empirical estimation of the computation required to discover approximate solutions to the traveling salesman problem using evolutionary programming," *Proc. of the Second Ann. Conf. on Evol. Prog.*, D. B. Fogel and W. Atmar, eds., pp. 56–61, Evolutionary Programming Society, La Jolla, CA.

[5] Yao, X., and Liu, Y. (1998) "Scaling up evolutionary programming algorithms," *Evolutionary Programming VII: Proc. of the Seventh Annual Conference on Evolutionary Programming (EP98), Lecture Notes in Computer Science*, vol. 1447, Springer-Verlag, Berlin, pp.103–112.

[6] Yao, X., Lin, G., and Liu, Y. (1997) "An analysis of evolutionary algorithms based on neighbourhood and step sizes," *Evolutionary Programming VI: Proc. of the Sixth*







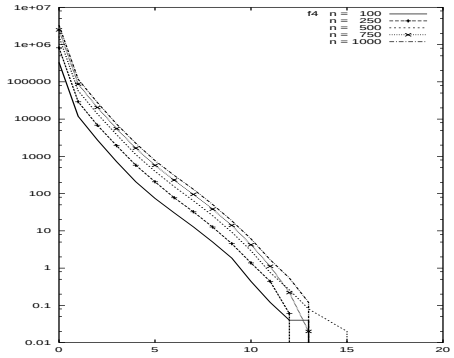
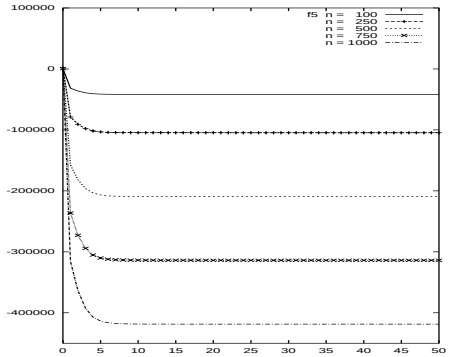
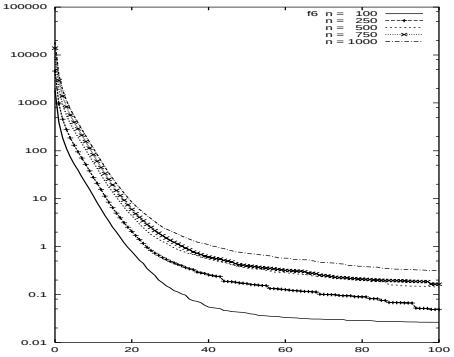
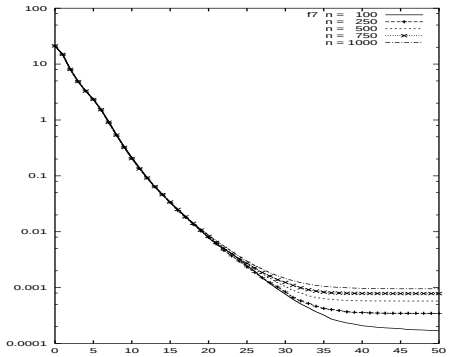


Figure 2: The evolution process of the mean best values found for f_1–f_4, with dimension $n = 100, 250, 500, 750$, and 1000, respectively, by FEPCC. The results were averaged over 50 runs. The vertical axis is the function value and the horizontal axis is the number of cycles. The function evaluation number in one cycle equals $100n$.







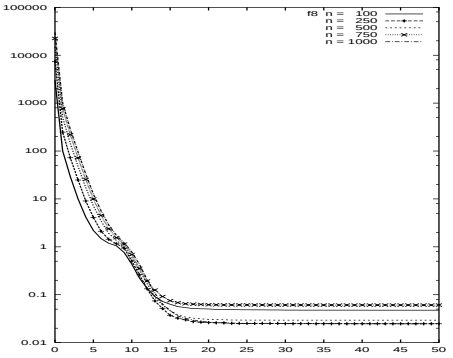
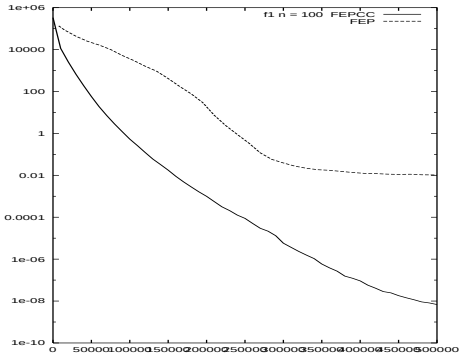
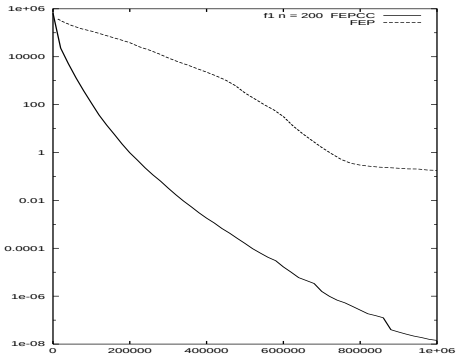


Figure 3: The evolution process of the mean best values found for f_5–f_8, with dimension $n = 100, 250, 500, 750$, and 1000, respectively, by FEPCC. The results were averaged over 50 runs. The vertical axis is the function value and the horizontal axis is the number of cycles. The function evaluation number in one cycle equals $100n$.





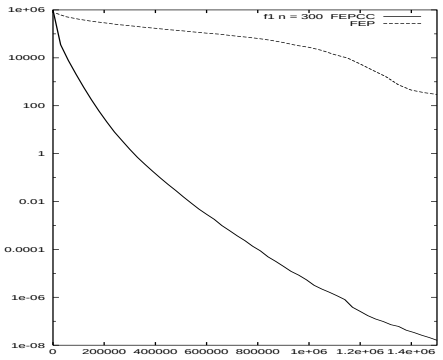
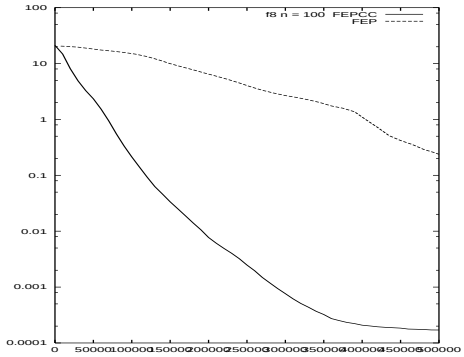
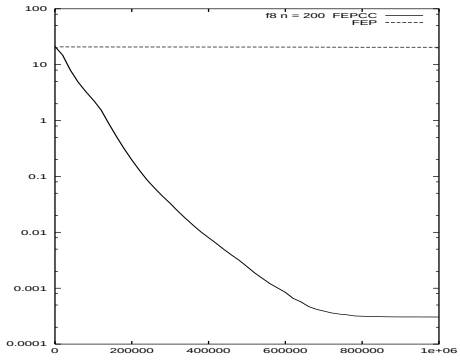


Figure 4: Comparison between FEPCC and FEP on f1 with dimension $n = 100$, 200, and 300, respectively. The results of FEPCC were averaged over 50 independent runs. The results of FEP were averaged over 10 independent runs [5].





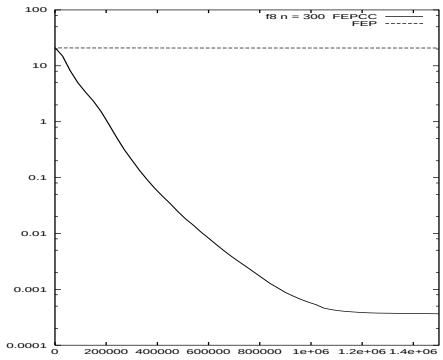


Figure 5: Comparison between FEPCC and FEP on f8 with dimension $n = 100$, 200, and 300, respectively. The results of FEPCC were averaged over 50 independent runs. The results of FEP were averaged over 10 independent runs [5].

Annual Conference on Evolutionary Programming, P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, eds., Lecture Notes in Computer Science, vol. 1213, pp. 297–307, Springer-Verlag.

[7] Potter, M. A., and De Jong, K. A. (1994) "A cooperative co-evolutionary approach to function optimization," in *Proc. of the Third International Conference on Parallel Problem Solving from Nature*, pp. 249–257, Springer-Verlag.

[8] Zhao, Q. F. (1998) "A general framework for cooperative co-evolutionary algorithms: a society model," *Proc. of 1998 IEEE International Conference on Evolutionary Computation*, pp.57–62.

[9] Bäck, T., and Schwefel, H.-P. (1993) "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation*, vol. 1, no. 1, pp. 1–23.

[10] Fogel, D. B. (1994) "An introduction to simulated evolutionary optimisation," *IEEE Trans. on Neural Networks*, vol. 5, no. 1, pp. 3–14.

[11] Feller, W. (1971) *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, Inc., 2nd ed.

[12] Törn, A., and Žilinskas, A. (1989) *Global Optimisation*, Lecture Notes in Computer Science, vol.350, Springer-Verlag, Berlin.

[13] Schwefel, H.-P. (1995) *Evolution and Optimum Seeking*, John Wiley & Sons, New York.

[14] Zhao, Q. F., Hammami, O., Kuroda, K., and Saito, K. (2000) "Cooperative co-evolutionary algorithm — how to evaluate a module ? " *Proc. the first IEEE Symposium on Combinations of Evolutionary Computation and Neural Networks*, pp. 150–157.