Concurrent Differential Evolution for Uncertain Optimization Problems

Kiyaharu Tagawa
School of Science and Engineering
Kinki University
Higashi-Osaka 577-8502, Japan
tagawa@info.kindai.ac.jp

Takashi Ishimizu
School of Science and Engineering
Kinki University
Higashi-Osaka 577-8502, Japan
takasi-i@info.kindai.ac.jp

***Abstract*—Multi-core CPUs, which have more than one processor (core), have been introduced widely into personal computers. Therefore, in order to utilize the additional cores to execute various costly application programs, concurrent implementations of them have been paid attention to. In this paper, a concurrent program of the latest evolutionary algorithm, i.e., differential evolution, is described. Furthermore, the concurrent program of differential evolution, which is called concurrent differential evolution, is revised to reduce the computational time for solving optimization problems in the presence of a wide range of uncertainties. Many real-world applications can be formulated as an uncertain optimization problem in which a probabilistic objective function has to be evaluated by using Monte Carlo integration. Consequently, it usually takes a long computational time to solve the uncertain optimization problem. The results of the numerical experiments conducted on two classes of uncertain optimization problems show that the revised version can reduce the computational time apparently comparing with the conventional version.**

Keywords-evolutionary algorithm; concurrent program; uncertain optimization problem; differential evolution.

I. INTRODUCTION

Differential Evolution (DE) is arguably one of the most powerful stochastic real-parameter optimization algorithms in current use [1], [2]. DE can be regarded as an evolutionary algorithm. However, comparing with typical evolutionary algorithms such as genetic algorithm, evolution strategy and particle swarm optimization, it has been reported that DE exhibits an overall excellent performance for a wide range of benchmark problems [3], [4]. Furthermore, because of its simple but powerful searching capability, DE has gotten numerous science and engineering applications [4].

The procedure of evolutionary algorithms for updating the individuals, i.e., tentative solutions, included in the population is called "generation alternation model". Evolutionary algorithms usually employ either of two types of generation alternation models [5]. The first one is called "generational model", while the second one is called "steady-state model". The original DE proposed by R. Storn and K. Price has been based on the generational model [1]. According to the generational model, DE holds two populations, namely the old one and the new one. After generating a new population, a concurrent population is replaced all together by the new population. On the other hand, a new DE based on the steady-state model has been proposed lately [6], [7]. The new DE is sometimes called "Sequential DE" (SDE) [6]. According to the steady-state model, SDE has only one population. Then each individual in the population is updated one by one. Comparing with the generational model, the steady-state model is usually suitable for parallelized evolutionary algorithms [10]. That is because evolutionary algorithms with the steady-state model need not synchronize the manipulation about any individual in each generation for replacing the old population by the new population.

Recently, multi-core CPUs have been introduced widely into personal computers. In order to utilize the additional cores to execute costly application programs, concurrent implementations of them are demanded [8]. In our previous paper [9], a concurrent program of DE, which is called Concurrent DE (CDE), was proposed. Exactly, the proposed CDE was a parallelized version of the above SDE.

In this paper, CDE is revised for solving uncertain optimization problems by using multi-core CPUs effectively. The revised CDE is called CDE for Uncertain optimization (CDEU). In many real-world optimization problems, a wide range of uncertainties have to be taken into account. Therefore, various evolutionary algorithms including DE for solving uncertain optimization problems have received increasing attention in recent years [12]–[14]. However, the conventional evolutionary algorithms applicable to uncertain optimization problems have not been parallelized for multi-core CPUs. Because the objective functions of uncertain optimization problems are approximated using Monte Carlo integration, the evaluation of them usually takes a very long time. CDEU is a promising optimizer for solving uncertain optimization problems with multi-core CPUs. The performance of CDEU is demonstrated in a comparison with CDE in two classes of uncertain optimization problems, namely noisy and robust optimization problems.

The rest of this paper is organized as follows. Section II describes SDE and a basic strategy of SDE used to generate a new individual. Section III describes CDE. Section IV explains uncertain optimization problems. Section V proposes CDEU for solving uncertain optimization problems. Section VI demonstrates the performance of CDEU in comparison with CDE. Finally, Section VII concludes this paper.

 ISBN: 978-1-61208-172-4

II. SEQUENTIAL DE (SDE)

In this section, we describe a new DE based on the steady-state model, i.e., SDE, and a basic strategy of SDE used to generate a new individual, i.e., a tentative solution.

A. Representation

The optimization problem can be formulated as shown in (1). The optimal solution of the optimization problem is represented by a D-dimensional real-parameter vector $\boldsymbol{x} = (x_0, \cdots, x_{D-1})$ that minimizes the value of the objective function $f(\boldsymbol{x})$. Besides, each $x_j \in \mathbb{R}$ is limited to the range between the lower $\underline{x}_j$ and the upper $\overline{x}_j$ boundaries.

$$\left[\begin{array}{ll} \text{minimize} & f(\boldsymbol{x}) = f(x_0, \cdots, x_j, \cdots, x_{D-1}) \\ \text{subject to} & \underline{x}_j \le x_j \le \overline{x}_j,\ j = 0, \cdots, D-1. \end{array}\right. \tag{1}$$

SDE is used to solve the optimization problem shown in (1). A tentative solution of (1) is represented by a real-parameter vector and called an "individual". SDE holds N_P individuals within the population for each generation. Therefore, the i-th individual $\boldsymbol{x}_i \in \mathbf{P}$ $(i = 0, \cdots, N_P - 1)$ arranged in the current population $\mathbf{P}$ is represented as

$$\boldsymbol{x}_i = (x_{0,i}, \cdots, x_{j,i}, \cdots, x_{D-1,i}) \tag{2}$$

where, $\underline{x}_j \le x_{j,i} \le \overline{x}_j,\ j = 0, \cdots, D-1$.

B. Strategy of SDE

In order to generate a new individual, SDE has borrowed the strategy of DE [6], [7]. In this paper, a basic strategy of DE named "DE/rand/1/exp" is described and used.

For each individual $\boldsymbol{x}_i \in \mathbf{P}$ $(i = 0, \cdots, N_P - 1)$, which is also called the target vector, three different individuals, say $\boldsymbol{x}_{r1}$, $\boldsymbol{x}_{r2}$ and $\boldsymbol{x}_{r3}$ $(i \neq r1 \neq r2 \neq r3)$, are selected randomly from the population $\mathbf{P}$. Then a new individual $\boldsymbol{u}$ called the trial vector is generated from them as shown in (3). An index $j_r \in [0,\ D-1]$ is selected randomly. As a result, the trial vector $\boldsymbol{u} = (u_0, \cdots, u_j, \cdots, u_{D-1})$ differs from the target vector $\boldsymbol{x}_i$ at least one element u_{j_r}. Besides, $\mathrm{rand}_j[0,1]$ denotes the random number generator that returns a uniformly distributed random number from within the range between 0 and 1. Both the scale factor S_F $(0 < S_F \le 1)$ and the crossover rate C_R $(0 \le C_R \le 1)$ are control parameters specified by the user in advance.

$$\left[\begin{array}{l} j = j_r; \\ \text{do } \{ \\ \quad u_j = x_{j,r1} + S_F \cdot (x_{j,r2} - x_{j,r3}); \\ \quad j = (j+1)\,\%\,D; \\ \} \text{ while } (\mathrm{rand}_j[0,1] < C_R \ \wedge\ j \neq j_r) \\ \text{while } (j \neq j_r)\ \{ \\ \quad u_j = x_{j,i}; \\ \quad j = (j+1)\,\%\,D; \\ \} \end{array}\right. \tag{3}$$

$$\left[\begin{array}{l} \text{Randomly generate } \boldsymbol{x}_i \in \mathbf{P}\ (i = 0, \cdots, N_P - 1); \\ \text{for } (i = 0;\ i < N_P;\ i{+}{+}) \text{ Evaluate } f(\boldsymbol{x}_i); \\ \text{for } (g = 0;\ g < G_M;\ g{+}{+})\ \{ \\ \quad \text{for } (i = 0;\ i < N_P;\ i{+}{+})\ \{ \\ \quad\quad \text{Generate } \boldsymbol{u} \text{ from } (3) \text{ and } (4); \\ \quad\quad \text{Evaluate } f(\boldsymbol{u}); \\ \quad\quad \text{if } (f(\boldsymbol{u}) \le f(\boldsymbol{x}_i))\ \{ \\ \quad\quad\quad \boldsymbol{x}_i = \boldsymbol{u}; \quad f(\boldsymbol{x}_i) = f(\boldsymbol{u}); \\ \quad\quad \} \\ \quad \} \\ \} \\ \text{Output the best } \boldsymbol{x}_i \in \mathbf{P}; \end{array}\right.$$

Figure 1. Pseudocode of SDE

If an element u_j of $\boldsymbol{u}$ comes out of the range $[\underline{x}_j,\ \overline{x}_j]$ by using the strategy in (3), it is returned to the range as

$$u_j = (\overline{x}_j - \underline{x}_j) \cdot \mathrm{rand}_j[0,1] + \underline{x}_j. \tag{4}$$

C. Procedure of SDE

Figure 1 shows the pseudocode of SDE. The stopping condition is specified by the maximum number of generations G_M. Since SDE is based on the steady-state model, only one population $\mathbf{P}$ is used. If a newborn trial vector $\boldsymbol{u}$ is not worse than the target vector $\boldsymbol{x}_i \in \mathbf{P}$, the target vector $\boldsymbol{x}_i$ is replaced by $\boldsymbol{u}$ immediately. Therefore, the excellent trial vector $\boldsymbol{u}$ can be used soon to generate offspring.

III. CONCURRENT DE (CDE)

A program is said to be concurrent if it can support two or more tasks in process at the same time. On the other hand, a program is said to be parallel if it can support two or more tasks executing simultaneously [8]. The difference between these definitions is the phrase in progress. The concurrent program performs multiple tasks in parallel if it runs on a multi-core CPU. Therefore, it can be expected that the execution time of an algorithm is reduced by using the concurrent program on the multi-core CPU.

The concurrent program of SDE is named Concurrent DE (CDE) [9]. Figure 2 shows the pseudocode of CDE. CDE employs a static allocation of tasks [8]. Therefore, the population $\mathbf{P}$ is divided equally into N_T chunks. Each chunk can be regarded as a subpopulation including N_P/N_T individuals. Then a task updating the individuals in one chunk is assigned to one thread statically. By using a multi-core CPU, one thread is executed by one core. Consequently, N_T tasks can be parallelized at the most by CDE.

CDE is based on MapReduce framework [11]. First of all, an initial population $\mathbf{P}$ is generated randomly. Then, in

 ISBN: 978-1-61208-172-4

```
Randomly generate x_i ∈ P (i = 0, ··· , N_P − 1);
// Map phase
for all n in parallel do {
   for (n = 0; n < N_T; n++) Thread(n);
}
Barrier();
// Reduce phase
Output the best x_i ∈ P;

Thread(n) {
   for (i = 0; i < N_P; i++) {
      if (i % N_T == n) {
         Evaluate f(x_i);
      }
   }
   for (g = 0; g < G_M; g++) {
      for (i = 0; i < N_P; i++) {
         if (i % N_T == n) {
            Generate u from (3) and (4);
            Evaluate f(u);
            if (f(u) ≤ f(x_i)) {
               x_i = u;    f(x_i) = f(u);
            }
         }
      }
   }
}
```

Figure 2. Pseudocode of CDE

```
Thread(n) {
   for (i = 0; i < N_P; i++) {
      if (i % N_T == n) {
         Evaluate F(x_i, N);
         Evaluate D(x_i, N);
      }
   }
   for (g = 0; g < G_M; g++) {
      for (i = 0; i < N_P; i++) {
         if (i % N_T == n) {
            Generate u from (3) and (4);
            Evaluate f(u);
            if (f(u) ≤ F(x_i, N) + α · D(x_i, N)) {
               Evaluate F(u, N);
               if (F(u, N) ≤ F(x_i, N)) {
                  x_i = u;    F(x_i, N) = F(u, N);
                  Evaluate D(u, N);
                  D(x_i, N) = D(u, N);
               }
            }
         }
      }
   }
}
```

Figure 3. Pseudocode of CDEU

Map phase, $\texttt{Thread}(n)$ ($n = 0, \cdots, N_T - 1$) are evoked. Each $\texttt{Thread}(n)$ method is assigned to one thread and contracts a task for updating all individuals included in one chunk. $\texttt{Barrier}()$ denotes the method that waits until all $\texttt{Thread}(n)$ methods have been completed. Finally, in Reduce phase, the best individual is selected from $\mathbf{P}$.

IV. Uncertain Optimization Problem

In many real-world optimization problems, a wide range of uncertainties have to be taken into account. Therefore, various evolutionary algorithms for solving uncertain optimization problems have received increasing attention in recent years [12]–[14]. Generally, uncertain optimization problems can be categorized into four classes. First, the objective function is noisy. Second, the decision variables may change after optimization, and the quality of the obtained optimal solutions should be robust against deviations from the optimal point. Third, the objective function is approximated, which means that the objective function suffers from approximation errors. Fourth, the optimum of the problem to be solved changes over time. In this paper, the first and the second classes of optimization problems are discussed.

A. Noisy Optimization Problem

The evaluation of the objective function is subject to noise. Noise in the evaluation of the objective function may come from many different sources such as sensory measurement errors or randomized simulations. Mathematically, a noisy objective function can be described as follows:

$$F(\boldsymbol{x}) = \int_{-\infty}^{\infty} (f(\boldsymbol{x}) + \delta)\, p(\delta)\, d\delta \tag{5}$$

where, $\boldsymbol{x}$ is a vector of decision variables and $f(\boldsymbol{x})$ is a time-invariant objective function considered in (1). δ is additive noise and $p(\delta)$ is the continuous density function of the noise δ. The noise δ is often assumed to be normally distributed with mean 0 and variance 1 such as $\delta = \mathcal{N}(0, 1)$.

In practice, the noisy objective function defined in (5) is approximated using Monte Carlo integration as

$$F(\boldsymbol{x}, N) = \frac{1}{N} \sum_{t=1}^{N} (f(\boldsymbol{x}) + \delta_t) \tag{6}$$

where, N is the number of random samples. The noise is given as follows: $\delta_t = \mathcal{N}(0, 1)$ $(t = 1, \cdots, N)$.

B. Robust Optimization Problem

The decision variables are subject to perturbations or changes after the optimal solution has been determined. Therefore, a common requirement is that a solution should still work satisfactorily when the decision variables change slightly. Such solutions are termed robust solutions. In order to search for robust solutions, a robust objective function can be described by using a vector of noises $\boldsymbol{\delta}$ as follows:

$$F(\boldsymbol{x}) = \int_{-\infty}^{\infty} f(\boldsymbol{x} + \boldsymbol{\delta})\, p(\boldsymbol{\delta})\, d\boldsymbol{\delta} \tag{7}$$

where, $\boldsymbol{\delta} = (\delta_0, \cdots, \delta_j, \cdots, \delta_D)$, $\delta_j = \mathcal{N}(0, 1)$.

In practice, the robust objective function defined in (7) is also approximated using Monte Carlo integration as

$$F(\boldsymbol{x}, N) = \frac{1}{N} \sum_{t=1}^{N} f(\boldsymbol{x} + \boldsymbol{\delta}_t) \tag{8}$$

where, $\boldsymbol{\delta}_t = (\delta_{0,t}, \cdots, \delta_{j,t}, \cdots, \delta_{D,t})$, $\delta_{j,t} = \mathcal{N}(0, 1)$.

V. CDE for Uncertain Optimization (CDEU)

CDE can be applied directly to uncertain optimization problems if the objective function $f(\boldsymbol{x})$ is replaced by the uncertain one $F(\boldsymbol{x}, N)$ defined by (6) or (8). However, in order to solve uncertain optimization problems more effectively, we propose a revised CDE. The revised CDE is called CDE for Uncertain optimization (CDEU). The procedure of CDEU is described in Figure 3. Since CDEU differs from CDE in `Thread`(n) part, only `Thread`(n) is shown in Figure 3. α is a control parameter. Besides, $D(\boldsymbol{x}, N)$ denotes a standard deviation of N sampled objective function values, namely $f(\boldsymbol{x}) + \delta_t$ or $f(\boldsymbol{x} + \boldsymbol{\delta}_t)$ $(t = 1, \cdots, N)$.

In order to reduce the evaluation time of the expensive $F(\boldsymbol{u}, N)$, CDEU prunes away the search for unpromising trial vectors estimated by using the time-invariant function $f(\boldsymbol{u})$. Even though the values of $f(\boldsymbol{u})$ and $D(\boldsymbol{u}, N)$ are required for CDEU, N sampled objective function values, $f(\boldsymbol{x}) + \delta_t$ or $f(\boldsymbol{x} + \boldsymbol{\delta}_t)$, evaluated for calculating $F(\boldsymbol{u}, N)$ can be reused for calculating $D(\boldsymbol{u}, N)$ again.

VI. Experiment

Through the numerical experiment conducted on two classes of uncertain optimization problems, the performance of the proposed CDEU is compared with that of CDE.

A. Test Function

For evaluating the performance of CDEU, the following four test functions are used as time-invariant ones. All the test functions have $D = 20$ dimensional real-parameters.

- Sphere function

$$f_1(\boldsymbol{x}) = \sum_{j=0}^{D-1} x_j^2$$
$$-100 \leq x_j \leq 100,\ j = 0, \cdots, D-1.$$

- Schwefel's function

$$f_2(\boldsymbol{x}) = \sum_{j=0}^{D-1} (\sum_{k=0}^{j} x_k)^2$$
$$-100 \leq x_j \leq 100,\ j = 0, \cdots, D-1.$$

- Rastrigin function

$$f_3(\boldsymbol{x}) = \sum_{j=0}^{D-1} (x_j^2 - 10\cos(2\pi x_j) + 10)$$
$$-5.12 \leq x_j \leq 5.12,\ j = 0, \cdots, D-1.$$

- Griewank function

$$f_4(\boldsymbol{x}) = \frac{1}{4000} \sum_{j=0}^{D-1} x_j^2 - \prod_{j=0}^{D-1} \cos\left(\frac{x_j}{\sqrt{j}}\right) + 1$$
$$-600 \leq x_j \leq 600,\ j = 0, \cdots, D-1.$$

B. Performance Criteria

In order to evaluate the performance of a parallel program for a multi-processor machine, the speedup is often used. The well-known definition of the speedup is the ratio of serial execution time to parallel execution time. However, in the assessment of a concurrent program, the conventional definition of the speedup has to be modified [8]. In the multi-core CPU, the executions of threads, which are evoked from a concurrent program, are assigned to respective cores automatically by operating system. Although the programmer can specify the number of threads in his program, he can not specify the number of cores used by his program.

Furthermore, in order to evaluate the performance of evolutionary algorithms such as CDE and CDEU, a single execution of them is not statistically significant. That is because the evolutionary algorithm is a stochastic algorithm. Therefore, a new speedup is defined as follows:

$$S(N_T, M) = \sum_{m=1}^{M} \frac{T_m(1)}{T_m(N_T)} \tag{9}$$

where, $T_m(N_T)$ $(m = 1, \cdots, M)$ is the execution time of CDE or CDEU achieved by using N_T threads. In an ideal situation, speedup is equal to the number of threads.

 ISBN: 978-1-61208-172-4

Table I
NOISY OBJECTIVE FUNCTION VALUE

F_p	F_1	F_2	F_3	F_4
CDEU	-0.242 (0.024)	-0.176 (0.038)	2.244 (1.165)	0.759 (0.034)
CDE	-0.284 (0.024)	-0.194 (0.034)	2.534 (1.439)	0.701 (0.030)

Table II
EXECUTION TIME FOR NOISY OPTIMIZATION PROBLEM

(a) Sphere function: F_1

N_T	1	2	4	6	8
CDEU	201 (10.8)	105 (8.3)	59 (10.6)	46 (5.0)	36 (7.5)
CDE	1085 (57.4)	565 (35.5)	306 (15.1)	244 (8.8)	189 (5.5)

(b) Schwefel's function: F_2

N_T	1	2	4	6	8
CDEU	216 (9.1)	117 (7.8)	64 (5.5)	52 (7.4)	43 (6.9)
CDE	1116 (55.5)	579 (36.6)	315 (18.8)	255 (8.9)	193 (10.4)

(c) Rastrigin function: F_3

N_T	1	2	4	6	8
CDEU	262 (11.6)	136 (10.8)	74 (6.8)	62 (8.4)	49 (5.6)
CDE	1185 (55.0)	614 (31.3)	329 (19.1)	265 (7.0)	208 (10.2)

(d) Griewank function: F_4

N_T	1	2	4	6	8
CDEU	285 (7.2)	149 (10.4)	79 (4.7)	65 (7.9)	53 (7.6)
CDE	1198 (56.0)	619 (32.5)	333 (19.3)	271 (10.3)	210 (12.6)

Table III
ROBUST OBJECTIVE FUNCTION VALUE

F_p	F_1	F_2	F_3	F_4
CDEU	18.21 (0.159)	146.6 (6.191)	209.9 (0.856)	0.793 (0.006)
CDE	18.20 (0.172)	145.1 (4.830)	210.5 (0.814)	0.791 (0.004)

Table IV
EXECUTION TIME FOR ROBUST OPTIMIZATION PROBLEM

(a) Sphere function: F_1

N_T	1	2	4	6	8
CDEU	17857 (127.6)	9016 (125.1)	4601 (132.7)	3908 (71.7)	2963 (75.1)
CDE	19516 (132.8)	9814 (139.6)	5009 (138.6)	4185 (84.1)	3190 (70.5)

(b) Schwefel's function: F_2

N_T	1	2	4	6	8
CDEU	13550 (289.1)	6870 (155.9)	3535 (127.9)	3195 (90.0)	2339 (82.8)
CDE	22434 (119.9)	11280 (120.6)	5795 (126.0)	4949 (78.0)	3772 (61.3)

(c) Rastrigin function: F_3

N_T	1	2	4	6	8
CDEU	24426 (198.2)	12295 (161.6)	6340 (133.0)	5612 (81.6)	4328 (48.9)
CDE	30424 (108.6)	15253 (91.7)	7858 (152.1)	6801 (61.6)	5191 (54.1)

(d) Griewank function: F_4

N_T	1	2	4	6	8
CDEU	24057 (308.2)	12103 (222.9)	6171 (175.4)	5297 (95.6)	3912 (72.2)
CDE	28688 (90.3)	14443 (220.7)	7320 (90.7)	6062 (60.1)	4623 (58.8)

C. Experimental Setup

CDEU and CDE were coded using the Java language, i.e., a popular language supporting multiple threads, and executed on a personal computer equipped with a multi-core CPU (CPU: Intel® Core$^{\text{TM}}$i7 @3.33[GHz]; OS: Microsoft Windows XP). The multi-core CPU has four cores that can respectively manipulate two threads at one time.

CDEU and CDE were applied respectively to some instances of uncertain optimization problems. During the experiments, the control parameters of them were fixed: the population size $N_P = 96$, the scale factor $S_F = 0.5$ and the crossover rate $C_R = 0.9$. For the stopping condition, the maximum number of generations was fixed to $G_M = 10^3$. Besides, $\alpha = 0.1$ is used for CDEU in all instances.

D. Result in Noisy Optimization Problem

CDEU and CDE were applied to four noisy optimization problems, in which the noisy objective functions $F_p(\boldsymbol{x}, N)$ were defined by (6) using the above test functions $f_p(\boldsymbol{x})$. The number of random samples was specified as $N = 100$. Table I shows the smallest noisy objective function values obtained by CDEU and CDE, which are averaged over $M = 20$ runs. The standard deviations of them also appear in parentheses. Similarly, Table II shows the execution times [ms] of CDEU and CDE using N_T threads. From Table I and Table II, CDEU has reduced the computational time without losing the quality of obtained solutions.

Figure 4 plots the speedups of CDEU achieved with N_T threads in the four noisy optimization problems. Similarly, Figure 5 plots the speedups of CDE. From Figure 4 and Figure 5, the speedups of CDEU and CDE have increased apparently with the increase in the number of threads.

E. Result in Robust Optimization Problem

CDEU and CDE were applied to four robust optimization problems in the same way with the noisy optimization problems. Table III compares the smallest robust objective function values, while Table IV shows the execution times. Figure 6 and Figure 7 also plot the speedups of CDEU and CDE achieved in the four robust optimization problems.

 ISBN: 978-1-61208-172-4

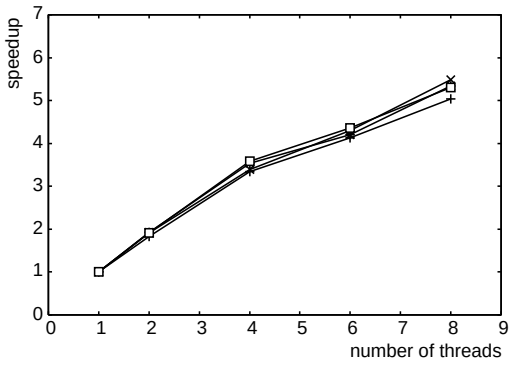

Figure 4. Speedups of CDEU in noisy optimization problems

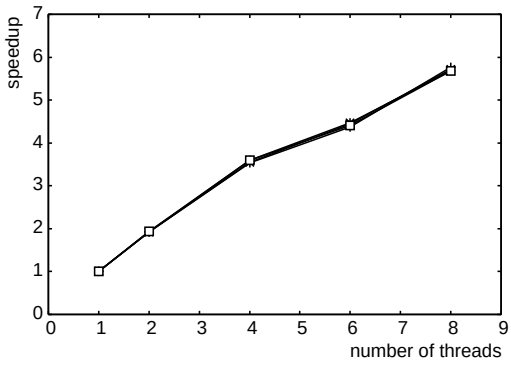

Figure 5. Speedups of CDE in noisy optimization problems

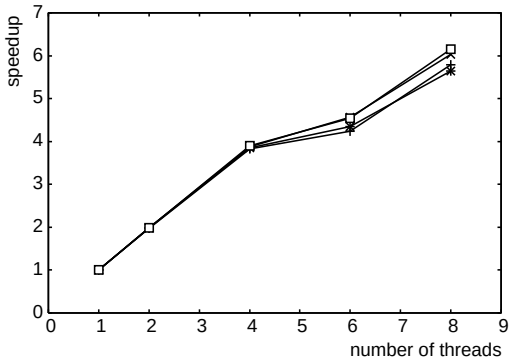

Figure 6. Speedups of CDEU in robust optimization problems

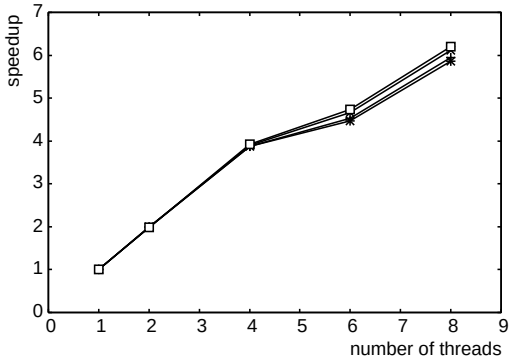

Figure 7. Speedups of CDE in robust optimization problems

VII. Conclusion

CDEU was proposed for the noisy and robust optimization problems. From the results of experiments, it was shown that CDEU could reduce the computational time comparing with CDE. Furthermore, CDEU could achieve the high speedup and the quality of obtained solutions as well as CDE.

In our future work, we would like to apply CDEU to real-world applications with various uncertainties.

Acknowledgment

The research was supported in part by the Grant-in-Aid for Scientific Research (C) (No. 21560432) from JSPS.

References

[1] R. Storn and K. Price, “Differential evolution - a simple and efficient heuristic for global optimization over continuous space,” *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[2] S. Das and P. N. Suganthan, “Differential evolution: a survey of the state-of-the art,” *IEEE Trans. on Evolutionary Computation*, vol. 15, no. 1, pp. 4–31, 2011.

[3] J. Vesterstrom and R. Thomson, “A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems,” *Proc. of IEEE Congress on Evolutionary Computation*, pp. 1980–1987, 2004.

[4] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*, Springer, 2005.

[5] G. Syswerda, “A study of reproduction in generational and steady-state genetic algorithms,” *Foundations of Genetic Algorithms 2*, Morgan Kaufmann Publ., pp. 94–101, 1991.

[6] V. Feoktistov, *Differential Evolution in Search Solutions*, Chapter 6, Springer, 2006.

[7] K. Tagawa, “A statistical study of the differential evolution based on continuous generation model,” *Proc. of IEEE Congress on Evolutionary Computation*, pp. 2614–2621, 2009.

[8] C. Breshears, *The Art of Concurrency - A Thread Monkey’s Guide to Writing Parallel Applications*, O’Reilly, 2009.

[9] K. Tagawa and T. Ishimizu, “Concurrent implementation of differential evolution,” *Proc. of WSEAS Int. Conference on Mew Aspects of System Theory and Scientific Computation*, pp. 65–70, 2010.

[10] B. D. Davison and K. Rasheed, “Effect of global parallelism on a steady state GA,” *Evolutionary Computation and Parallel Processing Workshop, Proc. of Genetic and Evolutionary Computation Conference Workshops*, pp. 167–170, 1999.

[11] J. Dean and S Ghemawat, “MapReduce: simplified data processing on large clusters,” *Proc. of 6th Symposium on Operating Systems Design and Implementation*, pp. 137–149, 2010.

[12] Y. Jin and J. Branke, “Evolutionary optimization in uncertain environments – a survey,” *IEEE Trans. on Evolutionary Computation*, vol. 9, no. 3, pp. 303–317, 2005.

[13] S. Das, A. Konar, and U. K. Chakraborty, “Improved differential evolution algorithms for handling noisy optimization problems,” *Proc. of IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1691–1698, 2005.

[14] H. G. Beyer and B. Sendhoff, “Evolution strategies for robust optimization,” *Proc. of IEEE Congress on Evolutionary Computation*, pp. 1346–1353, 2006.

 ISBN: 978-1-61208-172-4