A Structured Differential Evolutions for Various Network Topologies

Takashi Ishimizu and Kiyoharu Tagawa

***Abstract*—**A structured implementation of Differential Evolution (DE), which can be executed in parallel by using various networks topologies, is presented in this paper. Even though Evolutionary Algorithms (EAs) including DE have a parallel and distributed nature intrinsically, Sequential DE (SqDE) is especially suited for the structured implementation of DE. Therefore, the proposed Structured DE (StDE) is based on SqDE. Through the numerical experiment conducted on a variety of benchmark problems, the performances of StDE realized on some different network topologies are compared with the conventional SqDE that uses no networks. As a result, it is shown that the number of generations spent by StDE to find optimal solutions is smaller than the number of them spent by the above SqDE in many benchmark problems. Therefore, the optimal solutions of almost of the benchmark problems are found more efficiently by using the proposed StDE realized on the network topologies rather than SqDE.

***Keywords*—**Evolutionary Algorithm, Differential Evolution, Structured Differential Evolution, Parallel Algorithm

I. Introduction

Evolutionary algorithms (EAs) have been the subject of significant research in field of numerical optimization. Differential evolution (DE) is a new minimization method [1], capable of handling non-differentiable, non-linear and multimodel objective functions. DE has been designed as a stochastic parallel direct search method, that utilizes many practical concepts borrowed from the broad class of EAs, for solving real-parameter optimization problems. Comparing with typical EAs such as Genetic Algorithm (GA), Evolutionary Strategy (ES), and Particle Swarm Optimization (PSO), it has been reported that DE exhibits an overall excellent performance for a wide range of benchmark problems [2],[3]. Furthermore, because of its simple but powerful searching capability, DE has been applied to numerous real-world applications successfully [4]-[7].

The procedure of EAs for updating the individuals included in the population is called a “generation model” or a “generation alternation model”. EAs usually employ either of two types of generation models [5], [8]. The first one is called a “generational model” or a “discrete generation model”, while the second one is called a “steady-state model” or a “continuous generation model” [9]. The classic DE proposed originally by R. Storn and K. Price has been based on the discrete generation model [1]. According to the discrete generation model, the classic DE holds two populations, namely the old one and the new one. Then, by using a particular strategy, the individuals of the new population are generated from those of the old one. After that, the old population is replaced by the new one at a time.

Inspired by the great success of the classic DE, a variety of revised DEs have been developed for solving different types of optimization problems such as noisy [10], constrained [4], and multi-objective optimization problems [11],[12]. Furthermore, self-adaptive DEs that have various learning mechanisms to choose appropriate strategies and control parameters [13],[14],[27],[28]. However, many of the conventional DEs have been also based on the discrete generation model as well as the classic DE.

Recently, a new DE based on the continuous generation model is proposed [9],[15],[16]. The new DE is sometimes called “Sequential DE (SqDE)” [16]. According to the continuous generation model, SqDE holds only one population. Therefore, SqDE renews the individuals of the population one by one. SqDE generates a new individual called the “trial vector” from an existing individual called the “target vector” in the same way with the classic DE. After that, if the target vector included in the population is not better than the trial vector, the target vector is replaced by the trial vector immediately. Since the excellent newborn individual, namely the trial vector, can be used soon to generate offspring, it can be expected that SqDE finds good solutions faster than the classic DE [9].

Evolutionary Algorithms (EAs) including DE have a parallel and distributed nature intrinsically [19]. Therefore, various parallelization techniques of EAs have been proposed [19],[29]. Incidentally, parallel DE is also implemented by using Parallel Virtual Machine (PVM) [21]. However, throughout this paper, the structured EA is distinguished from the parallel EA. The structured EA consists of multiple structured populations connecting each other in accordance with a particular network topology. On the other hand, the parallel EA means every program of EA executed in parallel on multiple processors. Therefore, the structured EA can be realized by not only a single processor but also multiple processors connected by network.

In this paper, the structured DE (StDE) is proposed and evaluated in its performance. The StDE is one of a parallel implementation of SqDE, and uses multiple populations connected by some network topologies, namely the ring, the torus, the hypercube and so on. Through the numerical experiment, it is shown that the average of generations spent by StDE to find the optimal solutions using network topologies are smaller the average of them spent by SqDE not using any network topologies. Consequently, almost of optimal solutions are found more efficiently using network topologies.

II. Sequential DE (SqDE)

A. Representation

The optimal solution of the real-parameter 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, the value of each decision variable $x_j \in \Re$ is usually limited to the range between the lower $\underline{x}_j$ and the upper $\overline{x}_j$ boundaries. Therefore, the real-parameter optimization problem can be formulated as

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

Sequential Differential Evolution (SqDE) [16]is used to solve the optimization problem shown in (1). As well as conventional real-coded GAs [23] and DE [1], each tentative solution is represented by a real-parameter vector and called an "individual". Furthermore, DE holds N_P individuals within the population. Therefore, an individual $\boldsymbol{x}_i \ (i = 0, \cdots, N_P - 1)$ is represented as

$$\mathbf{x}_i = (x_{0,i}, \cdots, x_{j,i}, \cdots, x_{D-1,i})$$
$$\text{where, } \underline{x}_j \le x_{j,i} \le \overline{x}_j, \quad j = 0, \cdots, D-1.$$

Let rand[l,h] denote the random number generator which returns a uniformly distributed random number from within the range between l and $h \, (l, h \in \Re)$. The members of an initial population $\boldsymbol{x}_i \in P$ are generated randomly by using the random number generator as

```
for (i := 0; i < N_P; ++i) {
   for (j := 0; j < D; ++j) {
      x_{j,i} := rand[x_j, x̄_j];
   }
}
```

B. Strategy of DE

Differential mutation is a unique genetic operator of DE. Furthermore, a set of three genetic operators, namely, reproduction selection, differential mutation and crossover, is called the strategy of DE [1]. SqDE is also uses the strategy of DE [9], [15], [16]. Even though various strategies have been contrived for DE [3], four basic strategy named "DE/rand/1/bin", "DE/rand/1/exp", "DE/best/1/bin" and "DE/best/1/exp" are described and used in this paper. That is because those basic strategies are powerful enough for solving real-world application [3].

For each of the individuals $\boldsymbol{x}_i \ (i = 0, \cdots, N_P - 1)$ within the population, which is also called the target vector, three different individuals, say $\boldsymbol{x}_{\text{base}}, \boldsymbol{x}_{r1}$ and $\boldsymbol{x}_{r2} \ (i \ne \text{base} \ne r1 \ne r2)$, are selected from the current population. The individual $\boldsymbol{x}_{\text{base}}$ is called the base vector. In case of "DE/rand/1/bin" and "DE/rand/1/exp", the base vector and the other two individuals are selected randomly, on the other hand, in case of "DE/best/1/bin" and "DE/best/1/exp", the base vector is selected from the best vector among the population and the other two individuals are selected randomly.

Then a new individual $\boldsymbol{u}_i = (u_{0,i}, \cdots, u_{D-1,i})$ which is called the trial vector, is generated from the above four individuals through an assigned strategy. In case of "DE/rand/1/bin" and "DE/best/1/bin", the procedure of strategy is given by (2). On the other hand, in case of "DE/rand/1/exp" and "DE/best/1/exp", the procedure of the strategy is given by (3).

```
for( j := 0; j ≤ D−1;++ j){
  if (rand[0,1] < C_R ∨ j = j_r)
    u_{j,i} := x_{j,base} + S_F (x_{j,r1} − x_{j,r2});        (2)
  else u_{j,i} := x_{j,i};
}
```

```
j := j_r;
do {
  u_{j,i} := x_{j,base} + S_F (x_{j,r1} − x_{j,r2});
  j := ( j +1)%D;
} while (rand[0, 1] ≤ C_R ∧ j ≠ j_r);                        (3)
while ( j ≠ j_r) {
  u_{j,i} := x_{j,i};
  j := ( j +1)%D;
}
```

where $\boldsymbol{u}_i = (u_{0,i}, ..., u_{j,i}, ..., u_{D-1,i})$.

If an element of the trial vector $\boldsymbol{u}_i$ comes out of the range[$\underline{x}_j, \overline{x}_j$] by using the strategies shown in (2) and (3), it is returned to the range as:

$$u_{j,i} := \text{rand}[\underline{x}_j, \overline{x}_j].$$

In the strategies of DE shown in (2) and (3), the subscript $j_r \in [0, D-1]$ is selected randomly. Therefore, the trial vector $\boldsymbol{u}_i$ will be different from the target vector $\boldsymbol{x}_i$ at least one element. Besides the population size N_P, the scale factor $S_F \in (0, 1+]$ and the crossover rate $C_R \in [0, 1]$ are the control parameters of DE specified by the user in advance.

C. Procedure of SqDE

The procedure of the SqDE [16] can be described by using the following pseudo-code. Since SqDE is based on the continuous generation model, only one population $\boldsymbol{x}_i \in \boldsymbol{P}$ is used. If a newborn trial vector $\boldsymbol{u}_i$ is excellent, it is added to the population immediately. Therefore, in case of SqDE, the excellent trial vector $\boldsymbol{u}_i$ can be used soon to generate succeeding trial vectors.

[Pesudocode for SqDE]

```
Randomly generate x_i ∈ P;
for (i := 0; i < N_p; ++i)
   Evaluate f(x_i);
```

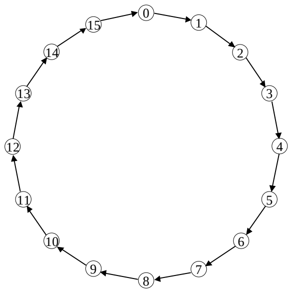

Fig. 1 Ring network

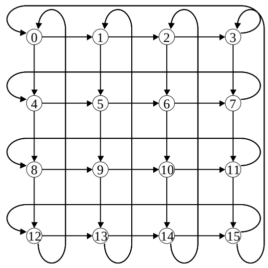

Fig. 2 Torus network

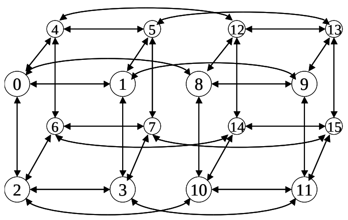

Fig. 3 Hypercube network

```
for (g := 0; g < G_M; ++g) {
  for (i := 0; i < N_P; ++i) {
    Generate u_i from (2) or (3);
    Evaluate f(u_i);
    if (f(u_i) ≤ f(x_i)) x_i := u_i;
  }
}
Output the best x_i ∈ P;
```

III. Structured DE (StDE)

A. Network Topology

For designing parallel or structured EAs, some network topologies are used. In network topologies, multiple population are connected mutually with some network topologies, namely, the ring, the mesh, the binary tree, the hypercube and so on. Besides, each processor can send messages to adjacent processors. In this paper, we use three network topologies, the ring, the torus and the hypercube.

Let Pr denote the number of processors. In case of the ring network, each processor P_p $(0 \le p < Pr)$ is connected to processor $P_{(p\text{-}1) \bmod Pr}$ and $P_{(p+1) \bmod Pr}$. Fig. 1 shows the ring network with 16 processors.

Let $P_{p,q}$ $(0 \le p.q < \sqrt{Pr})$ denote processor $P_{p\sqrt{Pr}+q}$. In case of the torus network, each processor $P_{p,q}$ is connected to processors $P_{(p-1) \bmod \sqrt{Pr},q}$, $P_{(p+1) \bmod \sqrt{Pr},q}$, $P_{p,(q-1) \bmod \sqrt{Pr}}$ and $P_{p,(q+1) \bmod \sqrt{Pr}}$ as shown in Fig. 2.

Let $\oplus$ denote binary operator that calculate exclusive OR for each bit. In case of the hypercube network, each processor P_p is connected to processors $P_{p \oplus 2^k}$ $(0 \le k < \log Pr)$. For example, processor P_5 (P_{0101}) is connected to processors P_4 (P_{0100}), P_7 (P_{0111}), P_1 (P_{0001}) and P_{13} (P_{1101}) in case of $Pr = 16$ as shown in Fig. 3.

B. Procedure of StDE

The procedure of the Structured DE (StDE) can described by using the following pseudo-code. In the StDE, we use two generation parameters g_l and g_s. g_l denotes the number of local generations. In the local generation, each processor executes SqDE for g_l times in parallel without communication each other. g_s denotes the number of super generations. Let $\boldsymbol{x}^{(p)}$ denote the best $\boldsymbol{x}_i \in \boldsymbol{P}$ at processor P_p. In the super generation, each processor executes the local generation and sends the best vector $\boldsymbol{x}^{(p)}$ to one of adjacent processors for g_s times. Incidentally, the procedure sending the best vector is called "migration", and g_l also denotes the migration frequency. As the stopping condition for StDE, the generations g_l and g_s are limited to the maximum numbers G_L and G_S respectively.

[Pesudocode for StDE]

```
Each P_p (0 ≤ p < Pr) executes in parallel {
  Randomly generate x_i ∈ P;
  for (i := 0; i < N_P; ++i)
    Evaluate f(x_i);
}
/* begin the super generation */
for (g_s := 0; g_s < G_S; ++g_s) {
  Each P_p (0 ≤ p < Pr) executes in parallel {
    /* begin the local generation */
    for (g_l := 0; g_l < G_L; ++g_l) {
      for (i := 0; i < N_P; ++i) {
        Generate u_i from (2) or (3);
        Evaluate f(u_i);
        if (f(u_i) ≤ f(x_i))
          x_i := u_i;
      }
    }
    /* end the local generation */
    /* begin the migration */
    Send x^(p) to one of the adjacent processors;
    Receive x^(q) from on of the adjacent processors P_q;
    Replace one of x_j ∈ P (x_j ≠ x^(p)) with x^(q);
    /* end the migration */
  }
}
/* end the super generation */
Each P_p (1 ≤ p < Pr) executes in parallel
  Send x^(p) to processor P_0;
P_0 receives (x^(p), ..., x^(Pr-1)) from other processors;
P_0 outputs the best x among (x^(p), ..., x^(Pr-1));
```

The end of each super generation, each processor P_p sends $\boldsymbol{x}^{(p)}$ to one of adjacent processors. The adjacent processor depends on the type of network topologies, namely the ring, the torus, the hypercube and the hierarchical network.

In case of the ring network, processor P_p $(0 \le p < Pr)$

sends $\boldsymbol{x}^{(p)}$ to the adjacent processor $P_{(p+1) \bmod Pr}$ and receives $\boldsymbol{x}^{(p-1) \bmod Pr}$ from processor $P_{(p-1) \bmod Pr}$ at each super generation.

In case of the torus network, processor $P_{p,q}$ $(0 \le p.q < \sqrt{Pr})$ sends $\boldsymbol{x}^{(p\sqrt{Pr}+q)}$ to adjacent processors $P_{p,(q+1) \bmod \sqrt{Pr}}$ if g_s mod 2 = 0, and processor $P_{p,q}$ sends it to adjacent processor $P_{(p+1) \bmod \sqrt{Pr},q}$ if g_s mod 2 = 1 at g_sth super generation.

In case of the hypercube network, processor P_p sends $\boldsymbol{x}^{(p)}$ to the adjacent processor $P_{p \oplus 2^{g_s \bmod \log Pr}}$ at g_sth super generation. For example, processor P_0 sends $\boldsymbol{x}^{(0)}$ to processors P_1, P_2, P_4 and P_8 at 1st, 2nd, 3rd and 4th super generation, respectively.

In case of the hierarchical network, which is also called the weighted hypercube network, processors are connected as hypercube, and P_p sends $\boldsymbol{x}^{(p)}$ to the adjacent processor $P_{p \oplus 2^{k \bmod \log Pr}}$ where $g_s \bmod 2^k = 0$ and $g_s \bmod 2^{k+1} \ne 0$. For example processor P_0 sends $\boldsymbol{x}^{(0)}$ to processors P_1, P_2, P_1, P_4, P_1, P_2, P_1, P_8 at 1st, ..., 8th super generations, respectively.

For comparative study, we also use the no networks. In case of the no networks, at the end of each super generation, each processor doesn't send $\boldsymbol{x}^{(p)}$. Namely, each processor executes the local generation for $G_L \times G_S$ times without communication.

IV. Numerical Experiment

A. Benchmark Problems

In order to evaluate the performance of StDE, the following nine benchmark problems are employed. f_1, f_2 and f_3 are unimodal functions, and f_4 ,... , f_9 are multimodal functions. f_1 and f_3 have D=16 dimensional real-parameters, and the other functions have D=8 dimensional real-parameters. Besides, the objective function values of their optimal solutions $\boldsymbol{x}^*$ are known as follows: $f_m(\boldsymbol{x}^*)=0$ (m=1, ..., 9).

- Sphere function (De Jong's 1st function)

$$f_1(\boldsymbol{x}) = \sum_{d=0}^{D-1} x_d^2,$$
$$-5.12 \le x_d \le 5.12, \quad d = 0, \cdots, D-1.$$

- Rosenbrock's function (De Jong's 2nd function)

$$f_2(\boldsymbol{x}) = \sum_{d=0}^{D-2} (100\,(x_{d+1} - x_d^2)^2 + (x_d - 1)^2),$$
$$-2.048 \le x_d \le 2.048, \quad d = 0, \cdots, D-1.$$

- Step function (De Jong's 3rd function)

$$f_3(\boldsymbol{x}) = \sum_{d=0}^{D-1} (\lfloor x_d \rfloor + 6),$$
$$-5.12 \le x_d \le 5.12, \quad d = 0, \cdots, D-1.$$

- Quartic function (De Jong's 4th function)

$$f_4(\boldsymbol{x}) = \sum_{d=0}^{D-1} (d x_d^4 + |\mathrm{Gauss}(0,1)|),$$
$$-1.28 \le x_d \le 1.28, \quad d = 0, \cdots, D-1,$$

where Gauss(0,1) denotes the Gaussian white noise. The Gaussian white noise makes sure that the algorithm doesn't get the same value on the same point.

- Shekel's function (De Jong's 5th function)

$$f_5(\mathbf{x}) = \frac{1}{0.002 + \sum_{i=0}^{24} \frac{1}{i + \sum_{d=0}^{D-1} (x_d - a_{d,i})^6}},$$
$$-65.536 \le x_d \le 65.536, \quad d = 0, \cdots, D-1,$$

the parameters for this function are:

$$a_{d,i} = \begin{cases} \{-40, -20, 0, 40\}, & \\ \quad \text{where } i = \{0,1,2,3,4\} & (\text{if } d \text{ is even}) \\ \quad \text{and } a_{d,i+5k} = a_{d,i}, k = \{1,2,3,4\} & \\ \{-40, -20, 0, 20, 40\} & \\ \quad \text{where } i = \{0,5,10,15,20\} & (\text{if } d \text{ is odd}) \\ \quad \text{and } a_{2d,i+k} = a_{d+1,i}, k = \{1,2,3,4\} & \end{cases}$$

- Rastrigin's function

$$f_6(\boldsymbol{x}) = \sum_{d=0}^{D-1} (x_d^2 - 10\cos(2\pi x_d) + 10),$$
$$-5.12 \le x_d \le 5.12, \quad d = 0, \cdots, D-1.$$

- Bohachevsky's function

$$f_7(\boldsymbol{x}) = \sum_{d=0}^{D-2} (x_d^2 + 2x_{d+1}^2 - 0.3\cos(3\pi x_d) - 0.4\cos(4\pi x_{d+1}) + 0.7),$$
$$-5.12 \le x_d \le 5.12, \quad j = 0, \cdots, D-1.$$

- Ackley's function

$$f_8(\boldsymbol{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{d=0}^{D-1} x_d^2}\right) - \exp\left(\frac{1}{D}\sum_{d=0}^{D-1}\cos(2\pi x_d)\right) + 20 + \exp,$$
$$-32.768 \le x_d \le 32.768, \quad d = 0, \cdots, D-1.$$

- Schaffer's function

$$f_9(\boldsymbol{x}) = \sum_{d=0}^{D-2} (x_d^2 + x_{d+1}^2)^{0.25} \times (\sin^2(50(x_d^2 + x_{d+1}^2)^{0.1}) + 1),$$
$$-100 \le x_d \le 100, \quad d = 0, \cdots, D-1.$$

B. Experimental Results about Strategies

StDE is coded by Java language, which is a very popular language supporting multiple threads, and executed on a personal computer equipped with a multi-core processor (CPU: Intel(R) Core^{TM} i7 @3.33[GHz]; OS: Microsoft Windows XP).

In order to evaluate the probability of finding the best solution, StDE are applied 256 times to the nine optimization problems f_1, ..., f_9 with five network topologies, namely the ring, the torus, the hypercube, the hierarchical network and the no networks simulated with 16 processors and with four strategies, namely "DE/rand/1/bin", "DE/rand/1/exp", "DE/best/1/bin" and "DE/ best/1/exp" respectively.

During the experiments, the following control parameters of every StDE are fixed: the population size N_P=32, the scale

Table I Average of generations to find the optimal solutions for benchmark problems

	networks	DE/rand/1/bin	DE/rand/1/exp	DE/best/1/bin	DE/best/1/exp
f_1	ring	554.1 (12.7)	290.0 (6.2)	329.5 (7.9)	245.6 (5.4)
	torus	562.9 (13.6)	289.3 (6.8)	333.2 (8.4)	245.9 (5.5)
	hypercube	567.0 (12.8)	289.0 (7.1)	335.2 (7.9)	246.1 (5.2)
	hierarchical	561.6 (11.9)	290.0 (8.2)	333.7 (6.2)	**245.0** (5.4)
	no networks	706.0 (16.8)	324.4 (6.8)	373.5 (10.4)	269.4 (6.3)
f_2	ring	1639.8 (62.1)	1208.9 (204.0)	962.1 (35.8)	852.6 (89.9)
	torus	1665.8 (53.8)	1195.3 (177.1)	995.3 (33.0)	**833.1** (101.1)
	hypercube	1668.1 (59.1)	1200.5 (187.2)	998.0 (35.5)	836.6 (93.9)
	hierarchical	1637.0 (44.6)	1177.3 (190.6)	990.8 (34.2)	842.8 (96.4)
	no networks	2350.7 (64.8)	2470.8 (213.8)	1238.8 (62.6)	1480.1 (165.7)
f_3	ring	842.0 (40.6)	231.8 (10.7)	471.2 (28.1)	**194.4** (8.0)
	torus	966.6 (48.3)	228.9 (11.7)	537.3 (31.3)	195.4 (7.8)
	hypercube	1039.3 (50.0)	230.6 (10.9)	558.8 (31.9)	195.3 (7.2)
	hierarchical	988.2 (47.6)	231.8 (11.5)	527.6 (33.6)	194.7 (8.2)
	no networks	1308.0 (61.2)	276.9 (11.0)	562.7 (36.8)	222.7 (9.2)
f_4	ring	466.5 (121.3)	449.8 (129.8)	357.3 (103.2)	414.5 (135.9)
	torus	467.2 (134.5)	470.2 (167.7)	356.1 (118.1)	414.0 (167.4)
	hypercube	470.7 (143.6)	437.6 (162.6)	**354.4** (124.5)	413.0 (176.9)
	hierarchical	467.0 (128.0)	454.7 (172.6)	370.3 (104.9)	418.6 (154.0)
	no networks	1860.2 (721.9)	1571.9 (614.2)	1135.3 (447.6)	1211.0 (475.6)
f_5	ring	643.6 (97.3)	413.3 (73.4)	426.5 (95.1)	**344.7** (94.5)
	torus	794.3 (106.9)	452.4 (72.2)	487.2 (80.3)	365.8 (81.4)
	hypercube	845.2 (130.6)	449.0 (69.6)	542.4 (92.9)	371.6 (63.1)
	hierarchical	794.9 (114.0)	439.6 (87.0)	504.7 (91.9)	355.2 (79.9)
	no networks	1732.2 (206.8)	935.0 (132.2)	620.2 (195.7)	523.9 (91.0)
f_6	ring	706.4 (50.5)	313.5 (13.0)	326.2 (21.3)	**217.8** (10.6)
	torus	794.7 (62.2)	320.8 (14.3)	348.8 (21.8)	228.7 (11.6)
	hypercube	818.3 (65.9)	324.0 (12.9)	355.2 (24.0)	227.8 (12.3)
	hierarchical	782.6 (60.7)	320.2 (13.8)	343.9 (22.7)	223.9 (11.3)
	no networks	872.2 (55.7)	339.5 (15.1)	318.2 (18.7)	220.8 (10.0)
f_7	ring	223.2 (7.1)	170.4 (5.7)	140.3 (4.7)	133.8 (4.7)
	torus	224.0 (6.6)	170.7 (5.7)	140.8 (4.7)	133.9 (4.7)
	hypercube	223.9 (7.1)	171.0 (5.3)	140.0 (5.0)	133.6 (4.2)
	hierarchical	223.5 (7.3)	170.2 (5.7)	140.1 (4.9)	**133.5** (4.6)
	no networks	245.4 (7.8)	184.8 (5.9)	149.2 (5.5)	142.7 (5.2)
f_8	ring	398.9 (7.7)	303.4 (5.8)	246.7 (5.2)	**236.8** (5.0)
	torus	400.4 (7.3).	304.2 (5.8)	246.8 (5.8)	237.2 (5.0)
	hypercube	400.2 (7.4)	304.4 (6.5)	247.0 (5.8)	237.6 (4.9)
	hierarchical	400.1 (7.4)	304.2 (5.8)	246.3 (5.8)	237.2 (4.7)
	no networks	410.6 (7.8)	317.4 (6.7)	248.6 (6.4)	244.4 (5.8)
f_9	ring	1141.3 (13.1)	775.0 (8.3)	690.5 (10.4)	**592.0** (6.7)
	torus	1152.5 (12.3)	780.2 (8.0)	695.9 (9.7)	594.8 (6.9)
	hypercube	1153.2 (11.0)	779.9 (7.7)	697.5 (8.8)	595.3 (6.4)
	hierarchical	1150.3 (12.6)	778.3 (8.3)	695.7 (9.5)	594.4 (5.7)
	no networks	1204.4 (21.0)	845.3 (13.0)	701.7 (14.2)	630.5 (10.1)

factor S_F=0.9 and the crossover rate C_R=0.5. As the stopping condition, the maximum generation is specified as G_L=8 and G_S =1024. As a result, the total number of generations becomes $G_L \times G_s = 8192$.

Table I shows the average of generations to find the optimal solutions with the ring, the torus, the hypercube, the hierarchical network, and the no networks for f_1, ..., f_9 respectively. The standard deviations of generations are also shown in the parentheses in Table I. From Table I, it is shown that the type of network topology doesn't much influence the number of generation to find the optimal solutions except in case of the no networks. Using any network topologies, almost of optimal solutions are found more efficiently than in case of the no networks. Therefore, we can conclude that the average of generations to find the optimal solutions is reduced using any processor networks. In addition, most of optimal solutions are found efficiently with the strategy "DE/best/1/exp" except f_4. That is because the average of generation to find the optimal solution is almost minimum in case of "DE/best/1/exp".

However, ``DE/best/1/exp'' is not always efficient. Table II shows the average generations to find the optimal solutions and probability of finding the optimal solutions at g=512, 1024 and 2048 with the ring network for f_5 (Shekel's function).

From the average of generations to find the optimal solutions, the strategy "DE/best/1/exp" is most efficient. On the other hand, from the probability of finding them at 1024th generation, the strategy "DE/rand/1/exp" is high probability to find, namely 245/256 = 95.7%.

In addition, the network topologies don't always work efficiently. Table III shows the average generations to find the optimal solutions and probability of finding them at g=512, 1024 and 2048 in case of "DE/rand/1/exp" for f_5. Considering the probability of finding the optimal solutions at 2048th generation, the optimal solutions are found in all trials, namely 256/256 = 100%, with the no networks.

Table II Probability of finding the optimal solutions at several generations with the ring for f_5

strategies	average generations	generations		
		g=512	g=1024	g=2048
DE/rand/1/bin	643.6	115	167	167
DE/rand/1/exp	413.3	222	245	**245**
DE/best/1/bin	426.5	93	93	93
DE/best/1/exp	344.7	176	183	183

(applied 256 times)

Table III Probability of finding the optimal solutions at several generations in case of DE/rand/1/exp for f_5

networks	average generations	generations		
		g=512	g=1024	g=2048
ring	413.3	222	245	245
torus	452.4	196	226	226
hypercube	449.0	202	231	231
hierarchical	439.6	217	251	251
no networks	935.0	0	184	**256**

(applied 256 times)

C. *Experimental Results about Migration Policies*

We consider the migration frequency of StDE. The end of the local generations, each processor sends the vest vector to another processor. Thus, in case of the number of the local generations g_l is small, namely in case of high migration frequency, fine-grained communication among processors is required. On the other hand, in case of g_l is large, namely in case of low migration frequency, StDE executes efficiently on the coarse grained parallel computing systems, such as BSP [30], CGM [31] and so on.

Fig. 4 shows the average generations to find the optimal solutions with the ring network for f_1 ,..., f_9 at each migration frequency. From these graphs in Fig.4, as the number of the local generations g_l is reduced, the average generations to find the optimal solutions are decreasing. Therefore, there is a trade-off relationship between migration frequency and the average generations.

Next, we notice the probability of finding the optimal solutions. The probabilities of finding the optimal solutions with the ring network for f_1, f_3, f_7 and f_8 are 100% regardless of the strategies or migration frequency. However, for the other five functions, f_2, f_4, f_5, f_6 and f_9, the probabilities of finding them are not always 100%. Table IV shows the probability of finding the optimal solutions with the ring network at 8192nd generation for f_2, f_4, f_5, f_6 and f_9, using the strategy "DE/best/1/exp". Note that in case of low migration frequency, namely in case of $g_l \le 4$, the average generations finding the optimal solutions are small, but the probabilities of finding them are not 100% even at the 8192nd generation. Therefore, to find the optimal solutions certainly, migration frequency should be enough high, namely, it should be $g_l > 8$.

Table IV Probability of finding the optimal solutions of several migration frequency with the ring at 8192nd generation.

	strategies	migration frequency				
		g_l=1	g_l=2	g_l=4	g_l=8	g_l=16
f_2	DE/rand/1/bin	254	256	256	256	256
	DE/rand/1/exp	247	251	256	256	256
	DE/best/1/bin	248	252	256	256	256
	DE/best/1/exp	240	244	256	256	256
f_4	DE/rand/1/bin	128	184	256	256	256
	DE/rand/1/exp	40	136	216	254	256
	DE/best/1/bin	152	124	256	256	256
	DE/best/1/exp	72	160	216	253	256
f_5	DE/rand/1/bin	30	62	121	167	230
	DE/rand/1/exp	23	49	131	245	256
	DE/best/1/bin	29	38	50	93	133
	DE/best/1/exp	21	40	89	183	248
f_6	DE/rand/1/bin	184	256	256	256	256
	DE/rand/1/exp	248	256	256	256	256
	DE/best/1/bin	216	256	256	256	256
	DE/best/1/exp	245	256	256	256	256
f_9	DE/rand/1/bin	256	256	256	256	256
	DE/rand/1/exp	232	256	256	256	256
	DE/best/1/bin	255	256	256	256	256
	DE/best/1/exp	248	256	256	256	256

(applied 256 times)

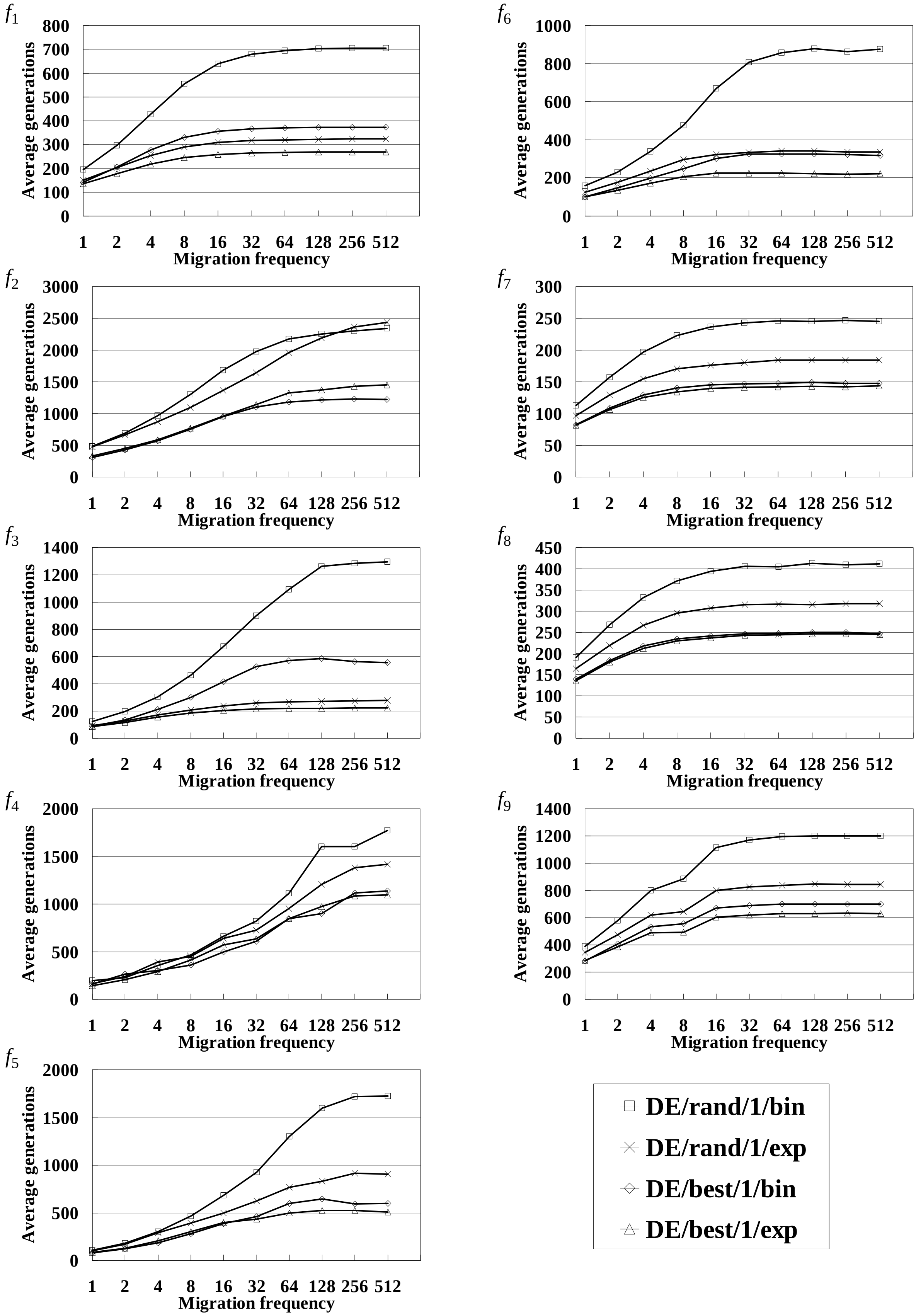

Fig.4 Performance of StDE for the benchmark problems

V. CONCLUSION

In this paper, the structured DE (StDE) is proposed and evaluated in its performance. The StDE is one of a parallel implementation of SqDE. The multiple populations of StDE are connected with some network topologies, namely, the ring, the torus and the hypercube. We show that the average of generations to find the optimal solutions using the network topologies, is smaller the average of them without the network topologies. Therefore, almost of the optimal solutions are found more efficiently using the processor networks. In addition, most of the optimal solutions are found efficiently with strategy “DE/best/1/exp”.

We also show that there is a trade-off relationship between the migration frequency and the average generations finding the optimal solutions. In case of high migration frequency, the average generations finding the optimal solutions are small, but the probabilities of finding them are not 100%.

In our feature work, we will evaluate the speedup of the proposed StDE on actual multi-processor system with some network topologies. Furthermore, we would like to apply the proposed StDE to real-world applications.

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

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

[4] R. Storn, “System design by constraint adaptation and differential evolution,” *IEEE Trans. Evolutionary Computation*, vol. 3, no. 1, pp. 22–34, 1999.

[5] C. Rotar, “Mutation Evolutionary Algorithm,” In Proc. *10th WSEAS Int. Conf. on Evolutionary Computing (EC '09),* 2009, pp.146–151.

[6] K. Tagawa, “Multi-objective optimum design of balanced SAW filters using generalized differential evolution,” *WSEAS Trans. System*, Issue 8, vol. 8, pp. 923–932, 2009.

[7] R. Oonsivilai and A. Oonsivilai, “Differential evolution application in temperature profile of fermenting process,” *WSEAS Trans. System*, Issue 6, vol. 9, pp. 618–628, 2010.

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

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

[10] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, “Opposition-based differential evolution for optimization of noisy problems,” in Proc. *IEEE Congress on Evolutionary Computation*, 2006, pp. 6756–6763.

[11] S. Kukkonen and J. Lampinen, “GDE3: The third evolution step of generalized differential evolution,” in Proc. *IEEE Congress on Evolutionary Computation*, 2005, pp. 443–450.

[12] U. K. Chakraborty, *Advances in Differential Evolution*. Springer, 2008.

[13] A. K. Qin and P. N. Suganthan, “Self-adaptive differential evolution algorithm for numerical optimization,” in Proc. *IEEE Congress on Evolutionary Computation*, 2005, pp. 1785–1791.

[14] J. Zhang and A. C. Sanderson, “JADE: Adaptive differential evolution with optional external archive,” *IEEE Trans. Evolutionary Computation*, vol. 13, no. 5, pp. 945–958, 2009.

[15] R.Thomsen, “Multimodal optimization using crowding-based differential evolution,” in Proc. *IEEE Congress on Evolutionary Computation*, 2004, pp.~1382–1389.

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

[17] K. Tagawa and H. Takada, “Comparative study of extended sequential differential evolutions,” in Proc. *the 9th WSEAS International Conference on Applications of Computer Engineering*, 2010, pp. 52–57.

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

[19] E. Alba and M. Tomassini, “Parallelism and evolutionary algorithms,” *IEEE Trans. Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.

[20] L. F. Bic and M. B. Dillencourt, “Advantages of self-migration for distributed computing,” *International Journal of Computers*, Issue 3, vol. 2, pp. 320–329, 2008.

[21] D. Zaharie and D. Petcu, “Parallel implementation of multi-population differential evolution,” *Concurrent Information Processing and Computing*, ISO Press, 2005, pp. 223–232.

[22] J. Dean and S. Ghemawat, “MapReduce: simplified data processing on large clusters,” in Proc. *6th Symposium on Operating Systems Design and Implementation*, 2004, pp. 137–149.

[23] L. J. Eshelman and J. D. Schaffer, “Real-code genetic algorithms and interval-schemata,” *Foundations of Genetic Algorithms 2*, Morgan Kaufmann Publ., 1993, pp. 187–202.

[24] K. Tagawa, “A comparative study of distance dependent survival selection for sequential DE,” in Proc. *IEEE International Conference on System, Man, and Cybernetics*, 2010, to be published.

[25] K-Y. Wong, Y-M. Choi, and S-W. Lam, “The design, implementation and application of the software framework for distributed computing,” *International Journal of Computers*, Issue 3, vol. 1, pp. 109–116, 2007.

[26] J. Wan, W. Yu, and X. Xu, “Design and implementation of distributed document clustering based on MapReduce,” in Proc. *the 2nd Symposium on International Computer Science and Computational Technology*, 2009, pp. 278–280.

[27] R.Gaemperle, S.D. Mueller and P. Koumoutsakos, “A Parameter Study for Differential Evolution,” 2002, in Proc.*3rd WSEAS International Conference on Evolutionary Computation (EC’02)*, 2002, pp. 4841–4846.

[28] J. Teo and M.Y. Hamid, “A Parameterless Differential Evolution Optimizer,” in Proc. *5th WSEAS/IASME International Conference on System Theory and Scientific Computation THEORY (ISTASC '05)*, 2005, pp.330–335.

[29] S. Selvi and D. Manimegalai, “Scheduling jobs on computational grid using Differential Evolution algorithm,” In Proc. *12th International Conference on Networking, VLSI and Signal Processing (ICNVS '10)*, 2010, pp.118–123.

[30] L.G.Valiant, “A bridging model for parallel computation,” *Communication of the ACM*, vol.33, no.8, 1990, pp.103–111.

[31] F.Dehne, A.Fabri and A.Rau-Chapman, “Scalable parallel computational geometry for coarse grained multicomputers,” in Proc. *ACM Symposium on Computational Geometry*, 1993, pp. 298–307.

Takashi Ishimizu received his M.E. and Ph.D. degrees from Nara Institute of Science and Technology (NAIST), in 1997 and 2000, respectively. He is now an Assistant Processor of the School of Science and Engineering, Kinki University Japan. His main researches are parallel algorithms and parallel complexity theory.

Kiyoharu Tagawa received his M.E. and Ph.D. degrees from Kobe University Japan, in 1993 and 1997, respectively. From 2005 to 2007, he served as an Associate Professor of the Faculty of Engineering, Kobe University. He is currently a Professor of the School of Science and Engineering, Kinki University Japan. His research interests include evolutionary computation, concurrent programming, and real-world applications.