Hybrid Integration of Differential Evolution with Artificial Bee Colony for Global Optimization

Bui Ngoc Tam[1], Pham Ngoc Hieu[1] and Hiroshi Hasegawa[2]

[1]*Graduate School of Engineering and Science, Shibaura Institute of Technology, Saitama, Japan*

[2]*College of Systems Engineering and Science, Shibaura Institute of Technology, Saitama, Japan*

Keywords: Artificial Bee Colony, Differential Evolution, Global Search, Hybrid Optimization Methods, Local Search, Multi-peak Problems.

Abstract: In this paper, we investigate the hybridization of a swarm intelligence algorithm and an evolutionary algorithm, namely, the Artificial Bee Colony (ABC) algorithm and Differential Evolution (DE), to solve continuous optimization problems. This Hybrid Integration of DE and ABC (HIDEABC) technique is based on integrating the DE algorithm with the principle of ABC to improve the neighborhood search for each particle in ABC. The swarm intelligence of the ABC algorithm and the global information obtained by the DE population approach facilitate balanced exploration and exploitation using the HIDEABC algorithm. All algorithms were applied to five benchmark functions and were compared using several different metrics.

1 INTRODUCTION

Two important areas of population-based optimization algorithms are swarm intelligence (SI) algorithms and evolutionary algorithms (EA). The search strategies used by living organisms have inspired the development of many optimization algorithms that are currently used by numerous engineering applications. One main concept in this approach has been to mimic how these organisms forage for food by using search agents to find a solution to a problem. A recent very successful algorithm in the SI class is the artificial bee colony algorithm (ABC) (Karaboga, 2005). ABC is simple and easy to implement, but it sometimes fails to find the global optimum in multi-peak or high steepness problems such as the Rastrigin or Rosenbrock functions. ABC searches only for the neighborhood of each employed bee or onlooker bee in one dimension during each iteration. Thus, the ABC algorithm component that makes employed bees or onlooker bees move to a new food source is too simple and this principle cannot cover the entire search range.

Another strategy is a natural EA known as Differential Evolution (DE), which is a population-based parameter optimization technique that was originally proposed by Storn and Price (Storn and Price, 1995; Storn and Price, 1997; Price, 1999). DE is based on the same the principle as GA where new individuals are generated by mutation and crossover, and the variance within the population guides the choice of new search points. DE is very powerful, but there is very limited theoretical understanding of how it works and why it performs well. However, DE may fall into local optima and have a slow convergence speed during the last stage of iterations.

To further improve the overall performance of ABC and DE, we propose a new hybrid strategy based on a combination of the DE and ABC algorithms. The aim of this work is to hybridize these two successful algorithms at a components level to benefit from their respective strengths. Therefore, our hybrid approach has the merits of both DE and ABC. Section 2 briefly introduces ABC and DE. Section 3 describes HIDEABC. Section 4 evaluates the performance of HIDEABC using five benchmark test functions and presents our experimental results. Our conclusions are given in Section 5.

2 REVIEW OF STANDARD ABC AND DE

2.1 Formulation of the Optimization Problem

The optimization problem is formulated in this section. The design variable, objective function, and constraint condition are defined as follows:

$$\textit{Design variable: } x = [x_1, ..., x_D] \quad (1)$$

$$\textit{Objective function: } f(x) \rightarrow \textit{Minimum} \quad (2)$$

$$\textit{Constrain condition: } x^{lb} \leq x \leq x^{ub} \quad (3)$$

where $x^{lb} = [x_1^{lb}, ..., x_D^{lb}]$, $x^{ub} = [x_1^{ub}, ..., x_D^{ub}]$, and D denote the lower boundary condition vectors, upper boundary condition vectors, and number of design variable vectors, respectively.

2.2 Artificial Bee Colony Algorithm (ABC)

ABC is a novel swarm intelligence (SI) algorithm, which was inspired by the foraging behavior of honeybees. ABC was first introduced by Karaboga in 2005 (Karaboga, 2005).

ABC is simple in concept, easy to implement, and it uses few control parameters, and hence, it has attracted the attention of researchers and has been used widely for solving many numerical (Karaboga and Basturk, 2007), (Karaboga and Basturk, 2006) and practical engineering optimization problems (Karaboga et al., 2007), (Baykasoglu and Ozbakr, 2007).

There are two types of artificial bees:
- **First,** the employed bees that are currently exploiting a food source.
- **Second,** the unemployed bees that are continually looking for a food source.

Unemployed bees are divided into scout bees that search around the nest and onlooker bees that wait at the nest and establish communication with employee bees.

The tasks of each type of bee are as follows:
- **Employed Bee:** A bee that continues to forage a food source that it visited previously is known as an employed bee.
- **Onlooker Bee:** A bee that waits in the dance area to make a decision about a food source is known as an onlooker bee.
- **Scout Bee:** When a nectar food source is abandoned by bees, it is replaced with new a food source found by scout bees. If a position cannot be improved further after a predetermined number of cycles, the food source is assumed to be abandoned. The predetermined number of cycles is an important control parameter for ABC, which is known as the "*limit*" before abandonment.

To better understand the basic behavioral characteristics of foragers, Karaboga (Karaboga, 2005) used figure 1. In this example, we have two discovered

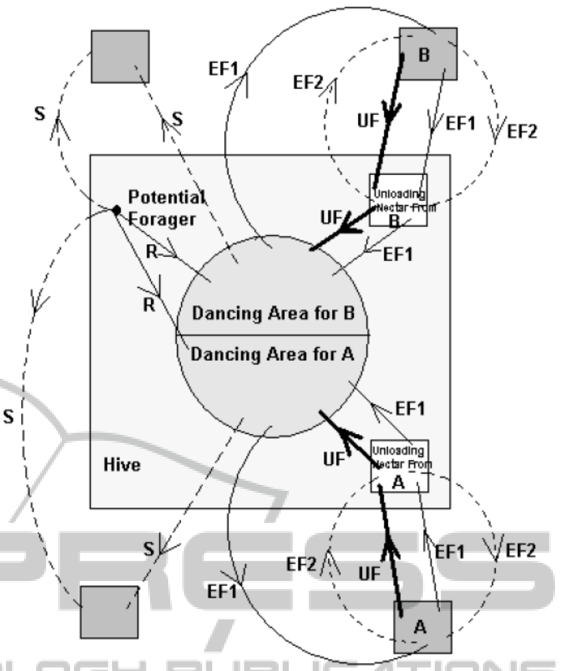

Figure 1: Behavior of honeybees foraging for nectar.

food sources: A and B. At the start, a potential forager is an unemployed forager. This bee will have no knowledge of the food sources around the nest. There are the following two possible options for this bee.
- The bee can become a scout and start searching around the nest spontaneously for a food source owing to some internal motivation or possible external clues (S in Figure 1).
- It can become a recruit after observing waggle dances and start exploiting a food source (R in Figure 1).

After locating the food source, the bee memorizes the location and immediately starts exploiting it. Thus, the bee will become an employed forager. The foraging bee collects a load of nectar from the source and returns to the hive, before unloading the nectar in a food store. After unloading the food, the bee has the following three options:
- It becomes a non-committed follower after abandoning the food source (UF).
- It dances and recruits nest mates before returning to the same food source (EF1).
- It continues to forage at the food source without recruiting other bees (EF2).

It is important to note that not all bees begin foraging simultaneously. Experiments have confirmed that new bees begin foraging at a rate proportional to the difference between the eventual total number of bees and the number that are currently foraging.

In the ABC algorithm, half of the colony consists of employed artificial bees while the other half consists of onlookers. For every food source, there is only one employed bee. In other words, the number of employed bees is equal to the number of food sources around the hive. An employed bee that exhausts its food source becomes a scout.

ABC algorithm simulation for optimization: In the ABC algorithm, the position of a food source i at generation G represents a possible solution to the optimization problem x_i^G, while the nectar amount in a food source corresponds to the quality (fitness fit_i^G) of the associated solution.

Algorithm 1: ABC Algorithm.
Requirements: Max Cycles, Colony Size, Limit.

```
Begin
1: Initialize the food sources
        $x_{i,j}^{G=0} = lb_j + rand_j * (ub_j - lb_j)$        (4)
   where $rand_j$ a random number in [0,1].
2: Evaluate the food sources
3: Cycle = 1
4: while $(Cycle \leq Max\_cycle)$ do
5: Produce new solutions using employed bees
        $v_{i,j} = x_{i,j} + \varphi_{i,j} * (x_{i,j} - x_{k,j})$        (5)
   where $k \in \{1,2,..,SN\}$ and $j \in \{1,2,..,D\}$ are randomly selected indices. Although $k$ is determined randomly, it has to be different from $i$. $\varphi_{ij}$ is a random number between $[-1,1]$. $v_{i,j}$ is the neighborhood of $x_{i,j}$ in dimension $j$.
6: Evaluate the new solutions and apply a greedy selection process
7: Calculate the probability values using the fitness values
        $p_i = \frac{fit_i^G}{\sum_{n=1}^{SN} fit_n^G}$        (6)
   where $fit_i^G$ is the fitness of food source $i$ at generation $G$.
8: Produce new solutions using onlooker bees
        $v_{i,j} = x_{i,j} + \varphi_{i,j} * (x_{i,j} - x_{k,j})$        (7)
9: Apply a greedy selection process for onlooker bees
10: Determine the abandoned solutions and generate new solutions randomly using scouts
        $x_{ij}^{G=0} = lb_j + rand_j * (ub_j - lb_j)$        (8)
   where $rand_j$ a random number in $[0,1]$.
11: Memorize the best solution found so far
12: $Cycle = Cycle + 1$
13: end while
14: return best solution
End
```

2.3 Differential Evolution (DE) Algorithm

DE was proposed by Storn and Price (Storn and Price, 1995) and is a very popular EA, which delivers remarkable performance with a wide variety of problems from diverse fields. Like other EAs, DE is a population-based stochastic search technique. It uses *mutation, crossover, and selection* operators at each generation to move its population toward the global optimum.

The DE technique combines simple arithmetic operators with the classical methods of crossover, mutation, and selection to evolve from a randomly generated starting population to a final solution.

At each generation G, DE creates a mutant vector $v_i^G = (v_{i,1}^G, v_{i,2}^G, ..., v_{i,D}^G)$ for each individual x_i^G(known as a target vector) in the current population. The five widely used DE mutation scheme operators are as follows (Wang, 2011).

DE/rand/1 scheme:

$$v_{i,j}^{G+1} = x_{r_1,j}^G + F(x_{r_2,j}^G - x_{r_3,j}^G) \quad (9)$$

DE /best /1 scheme:

$$v_{i,j}^{G+1} = x_{best,j}^G + F(x_{r_1,j}^G - x_{r_2,j}^G) \quad (10)$$

DE/target-to-best/1 scheme:

$$v_{i,j}^{G+1} = x_{i,j}^G + F((x_{best,j}^G - x_{i,j}^G) + (x_{r_1,j}^G - x_{r_2,j}^G)) \quad (11)$$

DE/best/2 scheme:

$$v_{i,j}^{G+1} = x_{best,j}^G + F((x_{r_1,j}^G - x_{r_2,j}^G) + (x_{r_3,j}^G - x_{r_4,j}^G)) \quad (12)$$

DE/rand/2 scheme:

$$v_{i,j}^{G+1} = x_{r_1,j}^G + F((x_{r_2,j}^G - x_{r_3,j}^G) + (x_{r_4,j}^G - x_{r_5,j}^G)) \quad (13)$$

In the above equations, r_1, r_2, r_3, r_4, and r_5 are distinct integers, which have been selected randomly from the range $[1,2,...,NP]$ and they are also different from i. The parameter F is called the scaling factor, which amplifies the difference vectors. x_{best}^G is the best individual in the current population.

After mutation, DE performs a binomial crossover operator on x_i^G and v_i^G to generate a trial vector $u_i^G = (u_{i,1}^G, u_{i,2}^G, ..., u_{i,D}^G)$

$$u_{i,j}^G = \begin{cases} v_{i,j}^G & \text{if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{i,j}^G & \text{otherwise} \end{cases} \quad (14)$$

where $i = 1,2,...,NP,\ j = 1,2,...,D,\ j_{rand}$ is a randomly chosen integer from $[1,D]$, $rand_j(0,\ 1)$ is a uniformly distributed random number between 0 and 1 that is generated for each j, and $CR \in [0,1]$ is the crossover control parameter. Because of the use of

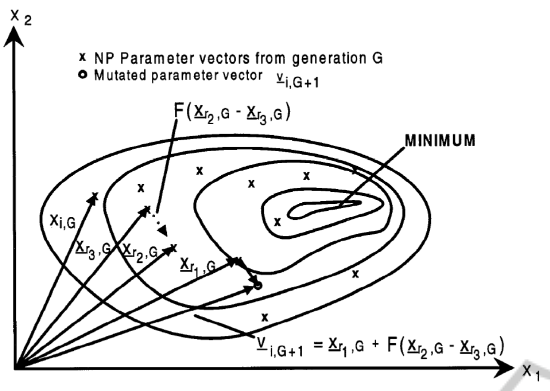
x 2
x NP Parameter vectors from generation G
o Mutated parameter vector $v_{i,G+1}$
$F(x_{r_2,G} - x_{r_3,G})$
MINIMUM
$x_{i,G}$
$x_{r_3,G}$
$x_{r_2,G}$
$x_{r_1,G}$
$v_{i,G+1} = x_{r_1,G} + F(x_{r_2,G} - x_{r_3,G})$
x 1

Figure 2: Example of a two-dimensional cost function showing its contour lines and the process of generating mutations of DE/rand/1.

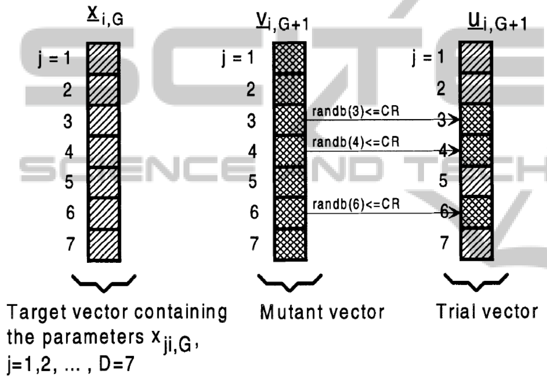

Figure 3: Illustration of the crossover process with $D = 7$.

j_{rand}, the trial vector u_i^G differs from its target vector x_i^G.
A selection operation is performed to choose whether the target vector x_i^G or the trial vector u_i^G should enter the next generation.

$$x_i^{G+1} = \begin{cases} u_i^G & \text{if } f(u_i^G) \leq f(x_i^G) \\ x_i^G & \text{otherwise} \end{cases} \tag{15}$$

The DE control parameters comprise the population size *NP*, the scaling factor *F*, and the crossover control parameter *CR*. Storn and Price (Storn and Price, 1995) argued that it is not difficult to set these three control parameters to obtain good performance. They suggested that *NP* should lie between 5*D* and 10*D*; a good initial choice for *F* is 0.5, whereas a value of *F* lower than 0.4 or higher than 1.0 will lead to performance degradation, and *CR* can be set to 0.1 or 0.9.

(Ronkkonen et al., 2005) suggested that *NP* should lie between 2*D* and 4*D*; *F* should be selected from the range [0.4, 0.95], with $F = 0.9$ being a good trade-off between convergence speed and robustness; and *CR* should lie between 0.0 and 0.2 for separable functions, and between 0.9 and 1.0 for multimodal and non-separable functions. Clearly, these researchers agreed that *F* should be in the range of [0.4, 1.0], and that *CR* should be close to 1.0 or 0.0, depending on the characteristics of problems.

Algorithm 2: DE Algorithm.

```
Requirements: Max Cycles, number of particles NP, crossover constant CR, and scaling factor F.
Begin
1: Initialize the population
       $x_{ij}^{G=0} = lb_j + rand_j * (ub_j - lb_j)$        (16)
   where $rand_j$ a random number in [0,1].
2: Evaluate the population
3: Cycle = 1
4: while ($Cycle \leq Max\_cycle$) for each individual $x_i^G$ do
5: Mutation: DE creates a mutation vector $v_i^G$ using equations (9) to (13), depending on the mutation scheme
6: Crossover: DE creates a trial vector $u_i^G$ using equation (14)
7: Greedy selection: To decide whether it should become a member of generation G + 1 (next generation), the trial vector $u_i^G$ is compared to the target vector $x_i^G$ (15)
8: Memorize the best solution found thus far
9: $Cycle = Cycle + 1$
10: end while
11: return best solution
End
```

3 HYBRID INTEGRATED DE AND ABC ALGORITHM (HIDEABC)

(Talbi, 2002) presented several hybridization methods for heuristic algorithms. According to (Talbi, 2002), two algorithms can be hybridized at a high level or low level using relay or co-evolutionary methods, which may be homogeneous or heterogeneous.

In this study, we hybridized ABC with DE using a low-level co-evolutionary heterogeneous hybrid approach. The hybrid is low-level because we combined the functionality of both algorithms. It is co-evolutionary because we did not use both algorithms in series, i.e., they run in parallel. Finally, it is heterogeneous because two different algorithms are used to produce the final results.

DE has many advantages, but it still has scope for improvement. In our test, we readily found that the DE did not always reach the best solution for a problem, because the algorithm sometimes reached a local optimum. To solve this problem, we propose a hybrid DE based on the ABC algorithm. In our proposed hybrid, we use the crossover and mutation principles of EA to enhance the neighborhood search of ABC.

The major difference between DE and ABC is how new individuals are generated. The new individuals generated during each generation are known as offsprings. The basic idea of HIDEABC is to combine the SI capacity of ABC with the local search capability of DE. In order to combine these algorithms, HIDEABC is proposed as follows. To generate new individuals in ABC, we use equations (9) to (13), depending on the mutation scheme, while equation (18) is used for DE crossover to enhance the neighborhood search of employed bees and onlooker bees.

The main steps of the HIDEABC algorithm are as follow:

- **Step 1:** In the first step, HIDEABC generates a randomly distributed initial population P($G = 0$) of *SN* solutions (food source positions), where *SN* denotes the size of the food source. Each solution (food source) x_i $(i = 1,2,...,SN)$ is a *D*-dimensional vector, where *D* is the number of optimization parameters.
- **Step 2:** Send the *employed bee* to the food sources and calculate their nectar amounts, before a new food source is found.
  In this process, the modification strategy uses the classical mutation and crossover components of the DE algorithm. This operation also improves the convergence speed and increases the diversity of the ABC population. We use *equations (9) to (13), depending on the mutation method, with eq(14) for crossover*.
- **Step 3:** An *onlooker bee* chooses a food source depending on the probability value associated with that food source, p_i: equation (7)
- **Step 4:** Send the *onlooker bees* to the food sources and determine their nectar amounts. Each onlooker evaluates the nectar information from all employed bees and selects a food source depending on the nectar amount available at the sources. The employed bee modifies the source position in her memory and checks its nectar amount. Provided that the nectar level is higher than that of the previous level, the bee memorizes the new position and forgets the old position.
  Use *equations (9) to (13), depending on the mutation method with equation (14) for crossover*.
- **Step 5:** A food source that is abandoned by bees is replaced with a new food source by the scouts. If a position cannot be improved further during a predetermined number of cycles *limit*, then that food source is assumed to be abandoned.
- **Step 6:** Send the scouts to the search area randomly to discover new food sources using equation (17).
- **Step 7:** Memorize the best food source found so far in the global best memory.
- **Repeat:** Steps 2 to 7 until the terminal requirements are met.

Algorithm 3: HIDEABC Algorithm.
Requirements: Max Cycles, number of particles NP, limit, crossover constant CR, and scaling factor F.
Begin
1: Initialize the food source positions

$$x_{ij}^{G=0} = lb_j + rand_j * (ub_j - lb_j) \tag{17}$$

where $rand_j$ is a random number in [0,1].
2: Evaluate the food sources
3: Cycle = 1
4: while $(Cycle \leq Max\_cycle)$ do
5: Produce new solutions using employed bees
Mutation process: Use equations (9) to (13), depending on the mutation scheme, to generate v_i^G
Crossover process:

$$u_{i,j}^G = \begin{cases} v_{i,j}^G & \text{if } rand_j(0,1) \leq CR \text{ or } j = j_{rand} \\ x_{i,j}^G & \text{otherwise} \\ x_{i,j}^G + \varphi_{i,j} * (x_{i,j}^G - x_{k,j}^G) & \text{if } j = para2change \end{cases} \tag{18}$$

where $i = 1,2,...,NP$, $j = 1,2,...,D$, j_{rand} and $para2change$ are randomly chosen integers from $[1,D]$, $rand_j(0,\ 1)$ is a uniformly distributed random number between 0 and 1 generated for each j, and $CR \in [0,1]$ is the crossover control parameter. Because of the use of j_{rand}, the trial vector u_i^G differs from its target vector x_i^G, and $k \in \{1,2,..,SN\}$ is a randomly chosen index $[1,NP]$. Although k is determined randomly, it should be different from i. φ_{ij} is a random number between [-1,1].
6: Evaluate the new solutions and apply a greedy selection process using equation (15).
7: Calculate the probability values based on their fitness values using equation (9).
8: Produce new solutions using onlooker bees; the process of neighborhood search by onlooker bees is the same as that used by employed bees.
9: Apply a greedy selection process to onlooker bees using equation (15).
10: Determine the abandoned solutions and generate new solutions randomly using scouts using equation

```
(11).
11: Memorize the best solution found so far
12: Cycle = Cycle + 1
13: end while
14: return the best solution
End
```

4 EXPERIMENTS

The first experiment tuned the F and CR parameters for DE and HIDEABC. Next, the F and CR parameters produced by the first test were used to compare the robustness of the optimization approach. These experiments involved 50 trials for each function. The initial seed number was varied randomly during each trial.

4.1 Benchmark Functions

To estimate the stability and convergence to the optimal solution using HIDEABC, we used five benchmark functions with 20 dimensions, namely, Rastrigin (RA), Ridge (RI), Griewank (GR), Ackley (AC), and Rosenbrock (RO). These functions are given as follows.

$$RA: f_1 = 10n + \sum_{i=1}^{n}\{x_i^2 - 10cos(2\pi x_i)\} \tag{19}$$

$$RI: f_2 = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2 \tag{20}$$

$$GR: f_3 = 1 + \sum_{i=1}^{n}\frac{x_i^2}{4000} - \prod_{i=1}^{n} cos\left(\frac{x_i}{\sqrt{i}}\right) \tag{21}$$

$$AC: f_4 = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e \tag{22}$$

$$RO: f_5 = \sum_{i=1}^{n}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \tag{23}$$

For each function, Table 1 lists the characteristics including the dependent terms, multi-peaks, and steepness. All of the functions are minimized to zero when the optimal variables $x_{opt} = 0$ and RO function $x_{opt} = 1$ are obtained. Note that it is difficult to search for optimal solutions by applying a single optimization strategy, because each function has specific complex characteristics.

Table 2 summarizes the design range variables. The search process is terminated when the search point reaches an optimal solution or a current generation process reaches the termination point.

Table 1: Characteristics of the benchmark functions.

Function	Dependent	Multi-peaks	Steepness
RA	No	Yes	Average
RI	Yes	No	Average
GR	Yes	Yes	Small
AC	No	Yes	Average
RO	Yes	No	High

Table 2: Design range variables of the benchmark functions.

Function	Design range
RA	$-5.12 \le x \le 5.12$
RI	$-51.2 \le x \le 51.2$
GR	$-51.2 \le x \le 51.2$
AC	$-5.12 \le x \le 5.12$
RO	$-2.048 \le x \le 2.048$

4.2 Tuning of F and CR Parameters for DE and HIDEABC

As mentioned above, the DE control parameters such as the population size NP, scaling factor F, and crossover control parameter CR are highly sensitive. In this test, we determined the best values of F and CR for each function using each approach. We tested 50 runs each for DE and HIDEABC. The population size was $NP = 8D = 160$, $F = 0.05, 0.1, \ldots, 0.95, 1.0$, $CR = 0.05, 0.1, \ldots, 0.95, 1.0$, and $Max\_cycle = 2500$. The experiment results are summarized in Table 3. The solutions of all benchmark functions reached their global optimum solutions.

4.3 Testing the Robustness of the Algorithms

4.3.1 Setting the Parameters for Standard ABC

The population size was $NP = 160$. The onlooker bees and employed bees constituted 50% each of the colony population, $SN = NP/2 = 80$, $FoodNumber = 80$, and $Limit = FoodNumber * D$.

4.3.2 Setting the Parameters for DE and HIDEABC

The population size was $NP = 160$. The F and CR values in Table 3 were used in the test, with the

Table 3: Best values for F and CR using 20 dimensions.

Function	RA		RI		GR		AC		RO	
	F	CR	F	CR	F	CR	F	CR	F	CR
DE/rand/1	0.10	0.15	0.55	0.95	0.15	0.65	0.15	0.65	0.45	0.95
DE/best/1	0.50	0.05	0.75	0.95	0.50	0.65	0.40	0.40	0.70	0.70
DE/target/1	0.50	0.05	0.65	0.95	0.55	0.85	0.45	0.50	0.65	0.65
DE/best/2	0.30	0.05	0.55	1.00	0.35	0.65	0.35	0.50	0.50	0.80
DE/rand/2	0.05	0.10	0.35	1.00	0.10	0.65	0.10	0.55	0.35	0.95
HIDEABC/rand/1	0.50	0.05	0.35	0.90	0.25	0.90	0.40	0.35	0.75	0.75
HIDEABC/best/1	0.50	0.05	0.35	0.90	0.30	0.90	0.50	0.50	0.65	0.65
HIDEABC/target/1	0.95	0.05	0.65	0.90	0.50	0.80	0.50	0.75	0.60	0.65
HIDEABC/best/2	0.50	0.05	0.65	1.00	0.20	0.90	0.35	0.55	0.55	0.75
HIDEABC/rand/2	0.10	0.10	0.50	1.00	0.15	0.60	0.15	0.55	0.40	0.95

same accuracy ($eps = 1.0e^{-6}$) to compare the iteration when the optimum was satisfied. If the success rate of the optimal solution was not 100%, ”–” is shown in Tables 4 to 8. We refer to HIDEABC/... as H/... for convenience.

Tables 4 to 8 show that the hybrid HIDEABC reached the global optimum in fewer iterations than with ABC and DE for all functions, with the exception of the RI function. The hybrid method failed to achieve better results with the RI function.

Tables 3 and 8 show that the results with HIDEABC/rand were good for the RA function, while Tables 7 and 8 show that they were good for the RI function. Tables 5, 6, and 7 show that the results were good with GR, that the AC function was good with HIDEABC/target/1, and that the RO function was good with HIDEABC/best/1. This problem depended on the function’s characteristics (see Table 1) and the control parameters F and CR.

Table 4: Number of iterations required to reach global optimum (Average results for 50 trials).

Function	ABC	DE/rand/1	H/rand/1
RA	543.10	349.62	**253.10**
RI	–	1670.80	**1079.00**
GR	246.84	106.26	**12.63**
AC	571.60	224.98	**78.60**
RO	–	1328.02	**712.72**

Table 5: Number of iterations required to reach global optimum (Average results for 50 trials).

Function	ABC	DE/best/1	H/best/1
RA	543.10	451.88	**311.50**
RI	–	815.98	1102.20
GR	246.84	55.56	**12.90**
AC	571.60	281.74	**79.80**
RO	–	1237.54	**592.69**

Table 6: Number of iterations required to reach global optimum (Average results for 50 trials).

Function	ABC	DE/target/1	H/target/1
RA	543.10	716.36	**398.43**
RI	–	581.68	1118.07
GR	246.84	59.34	**31.33**
AC	571.60	170.0	**73.00**
RO	–	1196.98	**539.52**

Table 7: Number of iterations required to reach global optimum (Average results for 50 trials).

Function	ABC	DE/best/2	H/best/2
RA	543.10	574.62	**407.03**
RI	–	546.94	841.30
GR	246.84	55.28	**12.23**
AC	571.60	148.46	**73.13**
RO	–	1049.56	**957.50**

Figures 5 to 9 show the average fitness of individual solutions until these methods reached the global optimum solutions. Numerical experiments showed that HIDEABC improved the generation number compared with the average generation results using simple ABC and DE. All benchmark functions reached their global optimum solutions. However, there were some differences among the methods. HIDEABC converged faster than ABC and DE, and it was particularly effective for the RO function.

Table 8: Number of iterations required to reach global optimum (Average results for 50 trials).

Function	ABC	DE/rand/2	H/rand/2
RA	543.10	329.72	**265.50**
RI	–	469.26	944.37
GR	246.84	104.52	**60.83**
AC	571.60	235.50	**135.70**
RO	–	1336.84	**856.80**

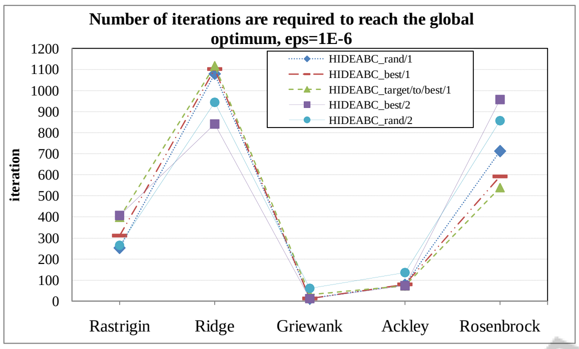

Figure 4: Number of iterations required to reach the global optimum using HIDEABC.

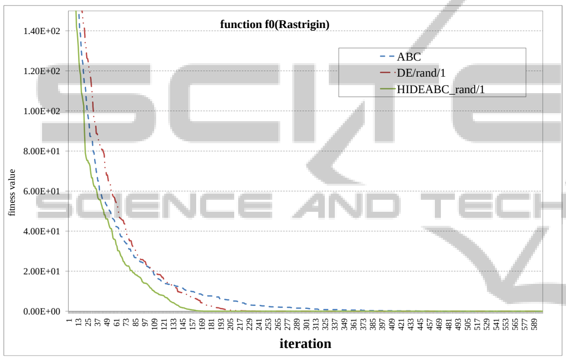

Figure 5: Convergence graph for the Rastrigin function.

HIDEABC arrived at the global optimum with a high probability for every function. In summary, this validation confirmed that the HIDEABC strategy can reduce the computational costs and improve the stability during convergence to the optimal solution.

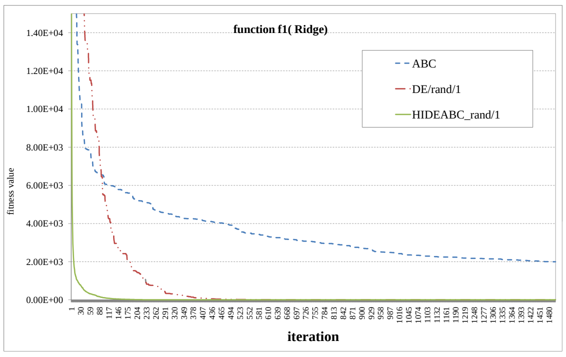

Figure 6: Convergence graph for the Ridge function.

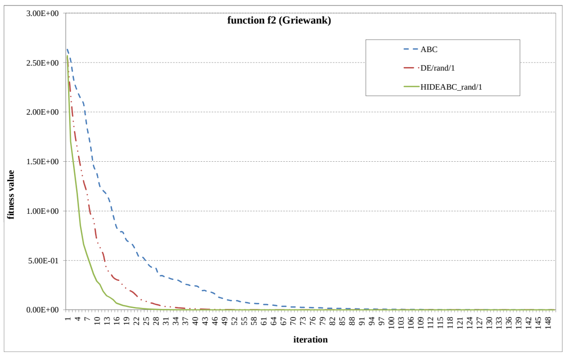

Figure 7: Convergence graph for the Griewank function.

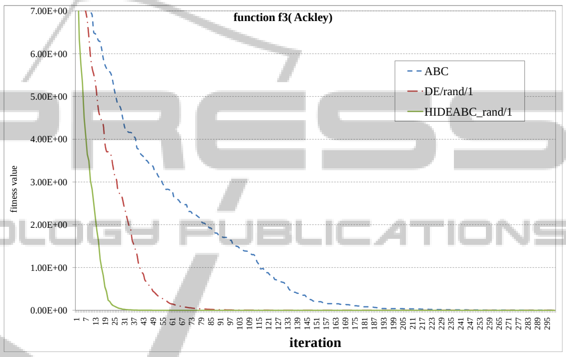

Figure 8: Convergence graph for the Ackley function.

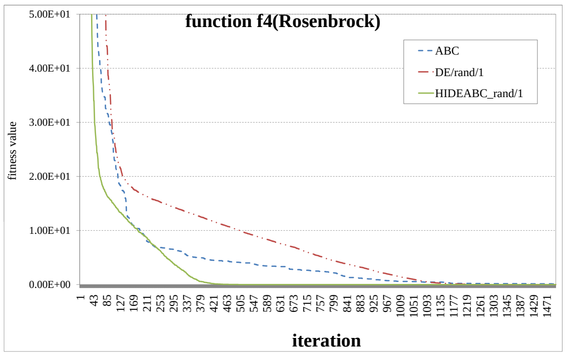

Figure 9: Convergence graph for the Rosenbrock function.

5 CONCLUSIONS

This paper introduces a new hybrid algorithm that exploits the strengths of ABC and DE. Our main concept is to integrate the exploitation capacities of ABC with the exploration abilities of DE. Five benchmark functions were used to validate the performance of HIDEABC compared with standard ABC and DE. The results showed that HIDEABC outperformed both tests for most functions. The results demonstrated that the convergence speed was faster with

HIDEABC than with ABC and DE.

In this study, we separated the F and CR parameters for each function in each method, and hence we had to tune these parameters. In future work, we will try to develop adaptive or self-adaptive F and CR. We confirmed that HIDEABC reduced the calculation costs and improved the time required for convergence to the optimal solution.

REFERENCES

Baykasoglu, A. and Ozbakr, L. (2007). Artificial bee colony algorithm and its application to generalized assignment problem, swarm intelligence: Focus on ant and particle swarm optimization. In *I-Tech Education and Publishing, Vienna, Austria*.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. In *TECHNICAL REPORT-TR06*.

Karaboga, D., Akay, B., and Ozturk, C. (2007). Artificial bee colony (abc) optimization algorithm for training feed-forward neural networks. In *Modeling Decisions for Artificial Intelligence*.

Karaboga, D. and Basturk, B. (2006). An artificial bee colony (abc) algorithm for numeric function optimization. In *IEEE Swarm Intelligence Symposium 2006, Indianapolis, Indiana, USA*.

Karaboga, D. and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization:artificial bee colony (abc) algorithm. In *Journal of Global Optimization*.

Price, K. (1999). An introduction to differential evolution. In *Corne, D., Dorigo, M., Glover, F. (eds.) New Ideas in Optimization*.

Ronkkonen, J., Kukkonen, S., and Price, K. (2005). Real parameter optimization with differential evolution. In *IEEE CEC, vol. 1*.

Storn, R. and Price, K. (1995). Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. In *Comput. Sci. Inst., Berkeley, CA, Tech. Rep. TR-95-012*.

Storn, R. and Price, K. (1997). Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. In *Journal of Global Optimization*.

Talbi, E. G. (2002). A taxonomy of hybrid metaheuristic. In *Journal of Heuristics*.

Wang, Y. (2011). Differential evolution with composite trial vector generation strategies and control parameters. In *IEEE Transactions on Evolutionary Computation*.