Information Sciences 182 (2012) 243–263

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Performance assessment of foraging algorithms vs. evolutionary algorithms

Mohammed El-Abd

Computer Engineering Department, American University of Kuwait, P.O. Box 3323, Safat 13034, Kuwait

ARTICLE INFO

Article history:
Received 14 December 2009
Received in revised form 1 September 2011
Accepted 5 September 2011
Available online 16 September 2011

Keywords:
Foraging algorithms
Evolutionary algorithms
Performance comparison
Evolutionary optimization

ABSTRACT

The class of foraging algorithms is a relatively new field based on mimicking the foraging behavior of animals, insects, birds or fish in order to develop efficient optimization algorithms. The artificial bee colony (ABC), the bees algorithm (BA), ant colony optimization (ACO), and bacterial foraging optimization algorithms (BFOA) are examples of this class to name a few. This work provides a complete performance assessment of the four mentioned algorithms in comparison to the widely known differential evolution (DE), genetic algorithms (GAs), harmony search (HS), and particle swarm optimization (PSO) algorithms when applied to the problem of unconstrained nonlinear continuous function optimization. To the best of our knowledge, most of the work conducted so far using foraging algorithms has been tested on classical functions. This work provides the comparison using the well-known CEC05 benchmark functions based on the solution reached, the success rate, and the performance rate.

1. Introduction

The foraging behavior of any living organism is defined as how this organism behaves in order to locate, handle, and ingest food. The approach taken in this behavior is referred to as the *search strategy*. For many animals, this strategy is usually broken down to three steps, namely: searching for and locating the prey, attacking the prey, and ingesting the prey [28]. The relative importance of these steps depends on the animal type and the environment characteristics.

Search strategies adopted by many living organisms inspired the development of different optimization algorithms currently adopted for various engineering applications. The main idea was for the search agents to imitate the foraging behavior of organisms in order to search for a solution of the problem. The algorithms included ABC [14] and BA [30] mimicking the foraging behavior of honey bees. BFOA [28], which mimics the foraging behavior of a swarm of E.coli bacteria, and ACO [10], which mimics the foraging behavior of ants.

Previous studies performed to assess the performance of some of these algorithms included the work in [20] showing that ABC performs better than PSO, an evolutionary algorithm (EA), and DE on a small suite of classical benchmark functions. The study in [17] compared ABC against PSO, a genetic algorithm (GA), DE, and an evolutionary strategy (ES) algorithm on a larger number of functions. It was shown that the performance of ABC is better than or at least similar to those algorithms while having a smaller number of parameters to tune. The work in [16] compared ABC to HS, and BA. The comparison was based on a small set of classical functions and ABC showed superior performance over both algorithms while producing reasonable results for higher dimensions.

The aforementioned studies suffer from two limitations. First, these studies only compare a certain algorithm to either evolutionary algorithms or other foraging algorithms but there is no single study that covers both. Second, the comparisons are based on a set of more or less classical functions. The proposed study overcomes these drawbacks by assessing the per-

E-mail address: melabd@auk.edu.kw

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.09.005

formance of a set of foraging algorithms against a set of evolutionary algorithms when applied to the well-known CEC05 benchmark functions [40]. The comparison is carried out based on three different metrics: namely, the solution reached, the success rate, and the performance rate. Although there are many improvements for these algorithms in the literature, the algorithms studied in this work are the basic versions.

This paper is organized as follows: Section 2 covers the studied algorithms and provides a conceptual component-based comparison between them. The experimental study is presented in Section 3. The paper is concluded in Section 4.

2. Studied algorithms

All the algorithms studied in this work are considered population-based algorithms, which is a class of meta-heuristics [7,42]. Population-based algorithms rely on iteratively updating a population of candidate solutions. The difference between foraging and evolutionary algorithms lies in the method adopted to update the population. It could be updated based on mimicking the foraging behavior of a living organism or by simulating an evolutionary process.

For foraging algorithms, ABC and BA simulate the foraging behavior of honey bees, BFOA simulates the foraging behavior of a swarm of E.coli bacteria, and ACO simulates the foraging behavior of ants. Although PSO could be considered a foraging algorithm, as it simulates the movement of a group of fish looking for food, to the best of our knowledge it has never been referred to as a foraging algorithm in the literature. The reason for this could be that PSO does not simulate the foraging behavior of a specific organism.

For evolutionary algorithms, the main approach is to simulate the evolutionary process in nature by adopting selection, crossover, mutation, and replacement operators. This class includes DE, GAs, ES, genetic programming (GP), and evolutionary programming (EP). Sometimes PSO is thought of as an evolutionary algorithm because the velocity and position update equations could be seen as a way of implementing the crossover and mutation operators.

Another type of population-based algorithms that is not tested in this work is the estimation of distribution algorithms (EDAs) [22]. The main difference is that instead of evolving a population of solutions, a probability model capturing the search space characteristics is evolved instead.

Some algorithms do not exactly fall under any category in this classification. One example is HS [23,41], which follows the concept of finding the right harmony for a musical instrument. In this sense, the algorithm neither follows the evolutionary approach nor mimics the foraging behavior of any living organism. Some see it as an evolutionary algorithm as what is important is having a harmony that is evolved somehow from one iteration to the next. On the other hand, some think of it as a foraging algorithm mimicking the foraging behavior of a musician searching for a harmony balance. Another example of such algorithms, although not tested in this work, is the gravitational search algorithm (GSA) [33], which is a population-based algorithm in which the individuals interact with each other based on Newtonian gravity and the laws of motion.

In the current study, we cover two algorithms that mimic the food foraging behavior of bees, for more of such algorithms one may refer to [6,18].

Finally, we would like to note that all the algorithms shown in this work are at a high level of abstraction. For more detailed algorithmic steps, the reader is encouraged to consult the given references.

2.1. Artificial bee colony

The ABC algorithm was first proposed in [14]. The algorithm was inspired by the method adopted of a swarm of honey bees to locate food sources. There are two different honey bee groups that share knowledge in order to successfully locate such sources. First, there are the *employed bees* that are currently exploiting a food source. Second, there are the *unemployed bees* that are continuously looking for a food source. Unemployed bees are divided into *scout bees* that search around the nest and *onlookers* that wait at the nest and establish communication with the employed bees.

This algorithm was applied to multidimensional and multi-modal function optimization in [14,19]. In ABC, S_n solutions to the problem are randomly initialized in the allowable domain and referred to as *food sources*. An employed bee is allocated to each food source and is used to find a new food source using the following equation:

$$v_{ij} = x_{ij} + \phi_{ij} \times (x_{ij} - x_{kj}), \tag{1}$$

where x_{ij} refers to problem variable j in the ith food source, j is a randomly selected number in $[1,D]$ and D is the number of dimensions, ϕ_{ij} is a random number uniformly distributed in the range $[-1,1]$, and k is the index of a randomly chosen solution. Both $\mathbf{v}_i$ and $\mathbf{x}_i$ are then compared against each other and the employed bee exploits the better food source, which is a *greedy selection* mechanism.

In the second step, each onlooker bee randomly chooses a food source according to the probability given in Eq. (2). Then, each onlooker bee tries to improve the selected food source using Eq. (1):

$$p_i = \frac{fit_i}{\sum_{j=1}^{S_n} fit_j}, \tag{2}$$

where fit_i is the fitness of the i^{th} food source.

Finally, if a certain food source i cannot be improved for a predetermined number of cycles, referred to as *limit*, this food source is abandoned. The employed bee that was exploiting this food source becomes a scout that looks for a new food source by randomly searching the problem domain using the following equation:

$$x_{ij} = lb_j + r \times (ub_j - lb_j), \tag{3}$$

where $j \in \{1 \dots D\}$ and D is the number of dimensions, lb_j and ub_j are the lower and upper bounds for problem variable j, and r is a random number uniformly distributed in the range $[0,1]$. The ABC algorithm is shown in Algorithm 1:

Algorithm 1. The ABC algorithm

Require: *Max_Cycles*, S_n, *limit*
1: Initialize the food sources
2: Evaluate the food sources
3: *Cycle*=1
4: **while** *Cycle* $\leqslant$ *Max_Cycles* **do**
5: Produce new solutions using employed bees as in Eq. (1)
6: Evaluate the new solutions and apply *greedy selection*
7: Calculate the probability using Eq. (2)
8: Produce new solutions using onlooker bees as in Eq. (1)
9: Evaluate the new solutions and apply *greedy selection*
10: **for** All solutions **do**
11: **if** A solution has not been improved for *limit* cycles **then**
12: Generate a new random solution using Eq. (3)
13: **end if**
14: **end for**
15: Memorize the best solution found so far
16: *Cycle* = *Cycle* + 1
17: **end while**
18: **return** best solution

2.2. The bees algorithm

Another algorithm that is inspired by the behavior of bees is the Bees algorithm (BA) proposed in [30]. In the harvesting season, a group of *scout bees* continuously search for food sources. After returning to the hive, the bees perform what is known as the *waggle dance* as a form of communication with other bees. The information contained in the dance helps the colony to send *flower bees* to these food sources precisely. In order for the colony to efficiently collect more food, more flower bees are sent to more promising food sources.

This is achieved in BA in a very simple form. The total number of food sources (solutions) investigated in an iteration is n, out of these n sources, the top m are referred to as the *best sites*. Out of the m best sites, the top e are referred to as *elite sites*. The elite sites, e, are exploited by *nep* bees while the rest of the best sites, $m - e$, are exploited by *nsp* bees, where *nep* > *nsp*. Any bee searches around a certain food source in a neighborhood of size *ngh* according to:

$$u_{ij} = x_{ij} + \phi_{ij} \times ngh, \tag{4}$$

for $i \in \{1 \dots m\}, j \in \{1 \dots D\}$ and D is the number of dimensions, and ϕ_{ij} is a random number uniformly distributed in the range $[-1,1]$. Solution $\mathbf{u}_i$ only replaces solution $\mathbf{x}_i$ if it has a better objective value.

Finally, the bottom $n - m$ bees are always re-initialized in the domain imitating the behavior of scout bees. These steps are shown in Algorithm 2.

Algorithm 2. The Bees algorithm

Require: *Max_Function_Evaluations*, *n*, *m*, *e*, *nsp*, *nep*, *ngh*
1: Initialize the food sources
2: Evaluate and sort the food sources
3: $Max\_Iterations = \frac{Max\_Function\_Evaluations}{n}$
4: *Iter_number*=1
5: **while** *iter_number* $\leqslant$ *Max_Iterations* **do**
6: **for** the top *e* sites **do**

(*continued on next page*)

Algorithm 2 (*continued*)

Algorithm 2. The Bees algorithm

```
    for nep bees do
      Exploit site using Eq. (4)
      Update site if the objective value has improved
    end for
  end for
  for the next m − e sites do
    for nsp bees do
      Exploit site using Eq. (4)
      Update site if the objective value has improved
    end for
  end for
  for the last n − m sites do
    Re-initialize the food source
  end for
  Evaluate and sort the food sources
  Iter_number = Iter_number + 1
end while
return best solution
```

2.3. Harmony search

HS [23,41] was introduced as an imitation of the musical process that searches for a harmony balance. In HS, each problem variable is a pitch of a different musical instrument and the complete solution is referred to as a harmony vector. If the pitch (decision variable value) makes a good harmony (good objective function value), it is stored in memory.

The memory part is modeled using a memory structure referred to as the Harmony Memory (HM) having a size *HMS*. Initially, HM is initialized with random harmonies. During the algorithm's progress, when a new harmony is found, it is inserted in HM replacing the worst harmony if it has a better objective function value.

In HS, new harmonies are developed by producing a series of new pitches. Each new pitch is produced following one of three rules. Playing one pitch from memory, picking one pitch from memory and adjusting it to play a new pitch, or playing a totally random pitch. These rules are applied using two parameters known as the Harmony Memory Considering Rate (*HMCR*) and the Pitch Adjusting Rate (*PAR*).

If the second rule is applied, a random pitch p_j is picked from memory and adjusted according to the following equation:

$$p_j = p_j + \phi \times bw, \tag{5}$$

for $j \in \{1 \ldots D\}$ and D is the number of dimensions, ϕ is a random number uniformly distributed in the range $[-1,1]$, and bw is an arbitrary *distance bandwidth*.

If a totally random new pitch is to be generated, it is generated randomly in the allowable domain using the following equation:

$$p_j = lb_j + r \times (ub_j - lb_j), \tag{6}$$

where lb_j and ub_j are the lower and upper bounds for decision variable j and r is a random number uniformly distributed in the range $[0,1]$. The complete steps are shown in Algorithm 3.

Algorithm 3. The HS algorithm

```
Require: Max_Function_Evaluations, memory size, HMCR, PAR, bw
Initialize the harmony memory
Evaluate the solutions in the memory
Max_Iterations = Max_Function_Evaluations
Iter_number = 1
while iter_number ⩽ Max_Iterations do
  for each dimension j do
    if U(0,1) ⩽ HMCR then
      Choose a randomly selected pitch from the memory
```

Algorithm 3 (*continued*)

Algorithm 3. The HS algorithm

```
    if U(0,1) ⩽ PAR then
      Adjust the selected pitch using Eq. (5)
    end if
  else
    Improvise a new pitch using Eq. (6)
  end if
end for
Evaluate the new harmony
if the new harmony is better than the worst one in memory then
  Replace the worst harmony in memory
end if
Iter_number = Iter_number + 1
end while
return best harmony
```

Many improvements exist for HS including the Improved HS (IHS) [24], the Global-best HS (GHS) [26], and the Differential HS (DHS) [8]. However, the algorithm tested in this work is a population-based harmony search that was proposed in [25]. In this work, we will refer to this algorithm as HS_{Pop}. In HS_{Pop}, the algorithm is exactly similar to Algorithm 3 with the exception of generating a population of harmonies in every iteration instead of a single one. Another modification is that *bw* is calculated as the standard deviation of the current population instead of being a randomly generated number.

2.4. Bacterial foraging optimization algorithm

BFOA [28] is inspired by the foraging behavior of a swarm of E.coli bacteria. The evolution of the swarm passes through four steps; namely, chemotaxis, swarming, reproduction, and elimination and dispersal as shown in Algorithm 4.

Chemotaxis are the motion patterns that the bacteria form in the presence of chemical attractants and repellents. The movement of the bacteria is achieved through tumbling and swimming. Tumbling is a one-step movement in a random direction while swimming is a continuous movement in a single direction. The bacteria alternate between the two modes of movement through their life cycle. A single chemotaxis step, where all the bacteria are moved, is maintained for N_c iterations. While moving, each bacterium either tumbles or swims. The swimming is only maintained if the bacterium is continuously moving in the direction of increasing nutrient concentration but to a maximum of N_s steps. For each bacterium θ_i, tumbling is defined as follows:

$$\theta_i^{t+1} = \theta_i^t + C(i) \times \frac{\triangle_i}{\sqrt{\triangle_i^T \triangle_i}}, \tag{7}$$

where t is the generation number, $C(i)$ is the step size (which could be different for each bacterium), and $\triangle_i$ is a random number in the range $[-1,1]$ chosen as a random direction for movement. Swimming use the same equation for tumbling while $\triangle_i$ is being fixed.

It was noted that groups of bacteria form swarms in semisolid nutrient media. Sometimes when under stress, bacteria release chemical substances that other bacteria can react to, resulting in the bacteria moving in groups protecting it from the stress. Swarming is simulated in BFOA by adding attractant/repellant signals between bacteria to the fitness function. For a bacterium θ_i, this effect is calculated by combining signals from all bacteria θ as follows:

$$J_{cc}(\theta_i,\theta) = \sum_{j=1}^{S} J_{cc}(\theta_i,\theta_j) = \sum_{j=1}^{S}\left[-d_{attract} \times e^{\left(-w_{attract}\sum_{m=1}^{D}(\theta_{jm}-\theta_{im})^2\right)}\right] + \sum_{j=1}^{S}\left[d_{repellant} \times e^{\left(-w_{repellant}\sum_{m=1}^{D}(\theta_{jm}-\theta_{im})^2\right)}\right] \tag{8}$$

where S is the swarm size, $d_{attract}$ and $w_{attract}$ ($d_{repellant}$ and $w_{repellant}$) are the depth of the attractant (repellant) released by the cell and width of the attractant (repellant) signal, and D is the size of the problem.

After every N_c chemotaxis steps, the bacteria reproduce. A healthy bacterium is split into two identical ones. This is simulated by selecting the best half of the swarm only to reproduce. Healthy bacteria are determined by the summation of their fitness over the period between the previous reproduction step and the current one.

In elimination and dispersal, which occurs every N_{re} reproduction steps, changes in the bacteria environment might cause sudden changes to the population. These changes might include the death of some of the bacteria or the movement from one region to another. This is simulated for each bacterium by eliminating then dispersing (re-initializing in the search space) it with a low probability p_{ed}.

Algorithm 4. The BFOA algorithm

Require: S, N_s, N_c, N_{re}, N_{ed}, p_{ed}, $C(i)$ *for* $i \in \{1 \ldots S\}$
Initialize the population
Evaluate the population
for N_{ed} elimination-dispersal steps **do**
  for N_{re} reproduction steps **do**
    for N_c chemotaxis steps **do**
      for every bacterium i **do**
        Move bacterium
        $m = 0$
        while fitness improves and $m < N_s$ **do**
          Move bacterium
          $m = m + 1$
        end while
      end for
    end for
    Determine healthy bacteria
    Reproduce
  end for
  for every bacterium i **do**
    if rand $< p_{ed}$ **then**
      Eliminate and disperse bacterium i
    end if
  end for
end for
return best solution

2.5. Ant colony optimization

An ACO approach for solving continuous optimization problems, referred to as ACO_R was proposed in [36]. The approach relies on estimating the joint probability distribution for one dimension at a time using mixtures of weighted Gaussian functions. This approach helps in identifying promising disjoint areas in the search space. The algorithm works by having a solution archive T of k solutions. For each solution s_t in the archive, the solution as well as its objective function value $f(s_t)$ are stored. This archive is used to define the univariate distributions. For each dimension d, the dimension is updated by sampling a Gaussian distribution selected from the archive for that dimension. The values of this dimension d across all the solutions in the archive compose the vector $\mu_{\mathbf{d}}$, which is the vector of means for the univariate Gaussian distributions.

$$\mu_{\mathbf{d}} = < s_{1d}, s_{2d}, \ldots, s_{kd} > . \tag{9}$$

To select one of these distributions, we have to calculate the weights vector $\mathbf{w}$, which holds the weights associated with these distributions. To calculate the weights, the solutions are ranked according to their fitness, with the best solution having a rank of 1. A weight is calculated for each solution (constant across all its dimensions) as follows:

$$w_l = \frac{1}{qk\sqrt{2\pi}} e^{-\frac{(l-1)^2}{2q^2k^2}}, \tag{10}$$

where q is a parameter that determines how much we should prefer good solutions, k is the number of solutions stored in the archive, and l is the solution rank.

To select a certain Gaussian function, a probability p_l is associated with each distribution (again constant across all its dimensions). This probability is calculated as the ratio of this function's weight to the summation of all weights:

$$p_l = \frac{w_l}{\sum_{r=1}^{k} w_r}. \tag{11}$$

After selecting a certain Gaussian function g denoted by its mean s_{gd}, where $1 < g < k$. The standard deviation for this functions is calculated as:

$$\sigma_{gd} = \xi \sum_{j=1}^{k} \frac{|s_{jd} - s_{gd}|}{k-1}, \tag{12}$$

where ξ is a parameter to balance the exploration–exploitation behaviors and is constant across all dimensions.

Finally the selected Gaussian function is used to produce m new solutions that are added to the archive while removing the worst m solutions. m is a parameter of the algorithm referred to as the number of ants. The whole process is shown in Algorithm 5.

Algorithm 5. The ACO_R algorithm

Require: *Max_Function_Evaluations*, *k*, *m*
1: Initialize the solutions in the archive *T*
2: Evaluate the solutions in the archive *T*
3: $Max\_Iterations = \frac{Max\_Function\_Evaluations}{m}$
4: *Iter_number* = 1
5: **while** *Iter_number* $\leqslant$ *Max_Iterations* **do**
6: Rank the solutions in the archive
7: Calculate the weights vector **w** using Eq. (10)
8: Calculate the probabilities vector **p** using Eq. (11)
9: **for** *m* ants **do**
10: Probabilistically select a Gaussian function
11: Calculate the standard deviation σ_{gd} using Eq. (12)
12: Generate the new solution
13: Add the new solution to the archive
14: **end for**
15: Remove the *m* worst solutions from the archive
16: *Iter_number* = *Iter_number* + 1
17: **end while**
18: **return** best solution in the archive

2.6. Differential evolution

DE [31,32,38] is a population-based algorithm in which the population is evolved from one generation to the next using mutation, crossover, and selection operators as shown in Algorithm 6.

For every individual $\mathbf{x}_i$ in the population, the individual is mutated by randomly choosing three different individuals other than the current one and setting the resultant vector (known as the *donor vector*) as the addition of the weighted difference of two of these individuals and the third. The mutation operator yields a donor vector $\mathbf{v}_i$, which is defined as follows (for the classical DE):

$$\mathbf{v}_i^t = \mathbf{x}_{r_1}^t + F \times \left(\mathbf{x}_{r_2}^t - \mathbf{x}_{r_3}^t\right), \tag{13}$$

where $i \in \{1 \ldots N\}$ and N is the number of individuals, t is the iteration number, r_1, r_2, and r_3 are three randomly selected indices in the range $[0,N]$ that are mutually exclusive and not equal to i, and F is a positive scaling factor.

Next, a crossover operation is carried between the individual $\mathbf{x}_i$ and the donor vector $\mathbf{v}_i$ to yield the trial vector $\mathbf{u}_i$. This operation is carried out according to a probability CR, which is defined as follows (for binomial crossover):

$$\mathbf{u}_{ij}^t = \begin{cases} \mathbf{v}_{ij}^t & \texttt{if} \quad (s_i \leqslant CR) \quad \text{or} \quad (j = rd_i), \\ \mathbf{x}_{ij}^t & \texttt{if} \quad (s_i > CR) \quad \text{or} \quad (j \neq rd_i). \end{cases} \tag{14}$$

where $j \in \{1 \ldots D\}$ and D is the number of dimensions, s_i is a randomly selected number in the range $[0,1]$, and rd_i is a randomly selected number in the range $[0,d]$ ensuring that at least one element from $\mathbf{v}_i$ is transferred to $\mathbf{u}_i$.

Finally, a selection step is performed to update $\mathbf{x}_i$ as follows:

$$\mathbf{x}_i^{t+1} = \begin{cases} \mathbf{x}_i^t & \texttt{if} \quad f\left(\mathbf{x}_i^t\right) < f\left(\mathbf{u}_i^t\right), \\ \mathbf{u}_i^t & \texttt{if} \quad f\left(\mathbf{x}_i^t\right) \geqslant f\left(\mathbf{u}_i^t\right). \end{cases} \tag{15}$$

Algorithm 6. The DE algorithm

Require *Max_Function_Evaluations*, *N*, *F*, *CR*
1: Initialize the population
2: Evaluate the population
3: $Max\_Iterations = \frac{Max\_Function\_Evaluations}{number\_of\_individuals}$
4: *Iter_number*=1
5: **while** *Iter_number* $\leqslant$ *Max_Iterations*
6: **for** every individual *i* **do**
7: Generate donor vector $\mathbf{v}_i$ using mutation in Eq. (13)

(*continued on next page*)

Algorithm 6 (*continued*)

Algorithm 6. The DE algorithm

```
    Generate trial vector u_i using crossover in Eq. (14)
    Update x_i using selection in Eq. (15)
  end for
  Memorize the best solution
  Iter_number = Iter_number + 1
end while
return best solution
```

In [34], the classical DE (utilizing the rand/1/bin strategy) defined above was tested against the CEC05 benchmark functions and the main remark was that the chosen limit of function evaluations was not enough for DE to achieve good results. It was stated that for a problem dimensionality of 10, DE needs about 100s of thousands to several millions of function evaluations to reach good solutions.

2.7. Particle swarm optimization

PSO is a population-based method, where the population is referred to as a *swarm*. The swarm consists of a number of individuals called *particles*. Each particle i in the swarm holds: (i) the current position $\mathbf{x}_i$, which represents a solution to the problem, (ii) the current velocity $\mathbf{v}_i$, (iii) the best position $\mathbf{pbest}_i$, the one associated with the best objective function value the particle has achieved so far, and (iv) the neighborhood best position $\mathbf{nbest}_i$, the one associated with the best objective function value found in the particle's neighborhood. The choice of $\mathbf{nbest}_i$ depends on the neighborhood topology adopted by the swarm, different neighborhood topologies have been studied in [21].

In traditional PSO, each particle adjusts its own position in every iteration in order to move towards its best position and the neighborhood best according to the following equations:

$$v_{ij}^{t+1} = w v_{ij}^{t} + c_1 r_1 (pbest_{ij}^{t} - x_{ij}^{t}) + c_2 r_2 \left(nbest_{ij}^{t} - x_{ij}^{t}\right), \tag{16}$$

$$x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}, \tag{17}$$

for $j \in \{1 \ldots D\}$ where D is the number of dimensions, $i \in \{1 \ldots n\}$ where n is the number of particles, t is the iteration number, w is the inertia weight, r_1 and r_2 are two random numbers uniformly distributed in the range $[0, 1)$, and c_1 and c_2 are known as the *acceleration factors*.

After changing its position, each particle updates its personal best position using (assuming a minimization problem):

$$\mathbf{pbest}_i^{t+1} = \begin{cases} \mathbf{pbest}_i^{t} & \texttt{if} \quad f\left(\mathbf{pbest}_i^{t}\right) \leqslant f\left(\mathbf{x}_i^{t+1}\right), \\ \mathbf{x}_i^{t+1} & \texttt{if} \quad f\left(\mathbf{pbest}_i^{t}\right) > f\left(\mathbf{x}_i^{t+1}\right). \end{cases} \tag{18}$$

Finally, the global best of the swarm is updated using the following equation:

$$\mathbf{gbest}^{t+1} = \arg \min_{\mathbf{pbest}_i^{t+1}} f\left(\mathbf{pbest}_i^{t+1}\right). \tag{19}$$

This model is referred to as the *lbest* (local best) model. Another simple model is the *gbest* (global best) model, which is the case when the particle's neighborhood is defined as the whole swarm. The basic PSO algorithm is shown in Algorithm 7.

Algorithm 7. The PSO algorithm

```
Require: Max_Function_Evaluations, n, w, c_1, c_2
Initialize the swarm
Evaluate the swarm
Max_Iterations = Max_Function_Evaluations / Num_Particles
Iter_number=1
while Iter_number ⩽ Max_Iterations do
  for every particle i do
    Update v_i using Eq. (16)
    Update x_i using Eq. (17)
    Update pbest_i using Eq. (18)
```

Algorithm 7 (*continued*)

Algorithm 7. The PSO algorithm

10: **end for**
11: Update **gbest** using Eq. (19)
12: *Iter_number* = *Iter_number* + 1
13: **end while**
14: **return gbest**

2.8. Comparing the different algorithms

In this subsection, a component-based comparison is carried between the different algorithms studied in this work.

2.8.1. The bee algorithms

Although ABC and BA both try to mimic the same bee behaviors, these behaviors are implemented differently:

- *Improving a food source:*
  Updating a bee in ABC involves cooperation between the different bees as the movement of a certain bee i involves the information gained from another randomly selected bee k. On the other hand, the update equation in BA performs a simple local search around the current solution with a fixed neighborhood size.
- *Exploring the better food sources:*
  Assigning more bees to better solutions is implemented in ABC through a probabilistic approach that is proportional to the fitness (thus the better the solution is, the larger the number of bees allocated to it). However, in BA, the number of elite sites is always fixed as well as the number of bees assigned to each site.
- *The scouting behavior:*
  Generating new random solutions is done differently in the two algorithms. In BA, the lower $n - m$ solutions are randomly re-initialized in every iteration while in ABC this does not happen unless a solution does not improve for a predetermined number of iterations. The approach adopted by ABC is better as it gives the chance for the bees to try to improve these solutions for a number of iterations before abandoning them.

2.8.2. Cooperation

An important behavior incorporated in most algorithms is *cooperation*. By cooperation we mean how individuals in the population can share information among each other. Algorithms involving cooperation are:

- *ABC:* The update equation for a bee takes into account the information provided by a another randomly selected bee.
- *HS:* The update of the current harmony might involve adjusting a pitch that is provided by another randomly selected harmony from memory.
- *DE:* Generating the donor vector uses information from another three different randomly selected individuals.
- *GA:* Cooperation is achieved through crossover.
- *PSO:* Each individual is influenced by the neighborhood best (or global best).

However, other algorithms do not involve such a behavior as the process of updating a single individual does not use information from other individuals:

- *BA:* Each individual is just updated through a simple neighborhood search.
- ACO_R: The solution produced by an ant is only based on a single solution from the archive.
- *BFOA:* Although the update equation for each individual is modeled as a swim down a promising direction, cooperation might be incorporated through the *swarming* behavior.

2.8.3. The re-initialization component

A component that ABC, BA, and BFOA share in common, which is not in ACO_R or PSO, is the ability to continuously generate completely random new solutions during the search (i.e. solutions that are not produced by the normal progression of the algorithm). This component is useful for exploration as it induces diversity in the population, delays convergence, and enables these algorithms to escape local minima. A disadvantage of such a component though might be in functions where the optimum lies outside of the initial bounds. As this component usually initializes new solutions within the initial bounds, it might drive the search away from the region containing the optimum.

However, this diversity inducing component is implemented differently in HS than all the other algorithms. HS only induces a limited number of randomly generated problem variables into a solution while in all the other algorithms complete random solutions are inserted into the population affecting the update equation for all individuals.

2.8.4. Steering the population

In most of the studied algorithms, the population evolves under influence from the best individuals. This is either done by direct communication (e.g. PSO), by allocating more search power to these regions (e.g. ABC, BA, and ACO_R), or by continuously selecting these individuals for mating or reproduction (GA and BFOA). Two algorithms however do not follow this paradigm, namely, DE and HS_{Pop}. Such a property could be helpful in preventing the population from quickly collapsing on such individuals.

3. Experimental study

3.1. Experimental setup

All the algorithms tested in this work are applied to the CEC05 benchmark functions [40] available at [39]. This library provides a class of shifted and/or rotated functions that are categorized into three different classes: uni-modal functions (f1–f5), multi-modal functions (f6–f14), and hybrid functions (f15–f25). For all experiments, the termination criterion is the maximum allowable number of function evaluations set as 100,000, 300,000, and 500,000 for dimensions 10, 30, and 50 respectively. For all the algorithms, initial candidate solutions are randomly initialized using uniform distribution over the specified domain. For each function, the reported solution is the average taken over 30 runs.

Performance assessment of the different algorithms is based on three metrics. First, the *best solution* reached after the allocated number of function evaluations; Second, the *success rate* defined as the number of successful runs over the total number of runs (a successful run is defined as a run where the tested algorithm has reached a predefined threshold for the function under study). Third, the *performance rate*, which is defined as:

$$Performancerate = \frac{FEV\_avg \times total\_runs}{successful\_runs}, \tag{20}$$

where *FEV_avg* is the average number of function evaluations needed to reach the predetermined threshold taken over the successful runs only. The threshold values are defined in [40] as 10^{-6} for uni-modal functions, 10^{-2} for multi-modal functions, and 10^{-1} for hybrid functions.

Experiments use the ABC code available at [15], the BA code available at [35], which is modified by the author to handle any problem size, HS code obtained by private communication with Geem , which is then modified according to the recommendations in [25], BFOA code available at [29], our implemented GA code, DE code available at [37], our implemented ACO_R code based on [36], and SPSO code (version 2007) available at [27]. Standard PSO has been verified by many researchers in the field and is very close to the original version of 1995 with a few improvements.

3.2. Parameter tuning

For ABC, the work in [1] indicated that there is no need to have a huge colony size in order to provide good results. The experiments were repeated using populations of 20, 40, and 100 bees for the three problem sizes. It was found that using a swarm of 40 bees provided the best results on average for all the dimensions. The recommendations in [17] were followed by setting the *limit* parameter to $S_n \times D$, although recent research [1] indicated that lower values might be needed for more difficult functions.

For BA, the number of food sources is set to 40, number of best sites $m = 30$, out of which there are $e = 10$ elite sites. There are $nep = 30$ bees exploiting elite sites and $nsp = 15$ bees exploiting rest of the best sites. In every iteration, the lower $n - m = 10$ food sources are re-initialized by scout bees. The neighborhood radius for the search is set as 1% of the search space width.

For HS_{Pop}, we followed the same parameter settings in [25] where $HMCR = 0.99$, $PAR = 0.5$, and bw was set as the standard deviation of the current population. The number of harmonies considered in every iteration is set to 40 as well as the value for *HMS* to have the same population size as the other algorithms.

For DE, the classical DE (rand/1/bin) [32] was used the same parameter settings as in [34] were followed. We set $F = 0.9$ and $CR = 0.9$. For separable functions, CR was set to 0.1. The population was set to 40 individuals. It was indicated in [34] that although the number of individuals should range between $4 \times D$ and $40 \times D$, setting it too high will not be of benefit given the allowable number of function evaluations. This was verified for the dimensionality of 30 by increasing the number of individuals to 60 and 120, but using 40 individuals only was still the best choice.

For BFOA, the drawback is having too many parameters to tune. Following the same parameter values in [28] provided very bad results in comparison with the other algorithms tested here. The reason for that as indicated in [28] might be that some of these values were biologically motivated but might not be the best for engineering applications. After extensive experiments, it was found that having a small population of only 10 bacteria and increasing both N_{re} and N_{ed} to 8 provided much better results. Other parameters were set as $N_c = 100$ and $N_s = 4$ while C is set as 0.5 for problems with large domains and 0.1 otherwise. The value of C has a great effect the performance, a recent introduction of an adaptive chemotaxis step could be found in [9]. Finally, the swarming behavior was not used in these experiments as it increased the computational cost while not providing a significant performance improvement. However, even if the swarming behavior is not used, BFOA

Table 1
Average solutions provided by all the algorithms for the uni-modal CEC05 functions taken over 30 runs.

Benchmark function	Size	ABC	BA	HS_{Pop}	BFOA	DE	GA	SPSO	ACO_R	IPOP-CMA-ES
f1	10	**0**	4.29e+03	**0**	9.82e−01	**0**	9.75e−04	**0**	1.33e−14	5.20e−09
		0	3.55e+03	**0**	2.44e−01	**0**	6.17e−04	**0**	2.45e−14	1.94e−09
f2		3.73	4.27e+03	54.93	3.62	7.53e−09	1.68	**0**	1.21e−13	4.70e−09
		2.52	4.09e+03	88.90	8.60e−01	8.73e−09	1.25	**0**	9.05e−14	1.56e−09
f3		6.41e+05	2.30e+07	2.77e+05	1.07e+05	**3.21e−01**	1.06e+06	5.64e+04	2.42e+06	*5.60e−09*
		2.78e+05	2.84e+07	3.07e+05	6.92e+04	**5.35e−01**	5.98e+05	3.27e+04	3.56e+06	*1.93e−09*
f4		29.17	1.01e+04	41.91	9.31e+03	2.35e−07	10.41	**0**	3.45e−13	5.02e−09
		41.83	4.39e+03	45.47	6.96e+03	2.16e−07	7.17	**0**	7.65e−13	1.71e−09
f5		86.76	7.76e+03	72.93	84.09	**0**	517.94	**0**	580.35	6.58e−09
		100.53	3.48e+03	191.19	22.04	**0**	339.48	**0**	732.51	2.17e−09
f1	30	1.06e−13	4.47e+04	2.31e−10	11.63	5.68e−14	3.59e−03	**0**	1.59e−13	5.42e−09
		1.97e−14	2.49e+04	1.28e−09	1.41	2.57e−29	1.61e−03	**0**	1.18e−13	9.80e−10
f2		2.49e+03	4.43e+03	891.77	155.95	172.10	97.15	**0**	3.24e−02	6.22e−09
		969.89	2.46e+04	178.81	32.55	48.33	40.63	**0**	8.61e−02	8.95e−10
f3		6.22e+06	2.14e+08	2.90e+06	2.67e+06	3.24e+06	6.36e+06	**2.21e+05**	2.88e+08	*5.55e−09*
		1.67e+06	2.25e+08	1.06e+06	1.19e+06	1.69e+06	2.42e+06	**9.53e+04**	2.25e+08	*1.09e−09*
f4		1.49e+04	7.60e+04	219.23	9.81e+04	556.24	2.49e+03	**8.35e−04**	1.01e+03	1.11e+04
		5.15e+03	2.15e+04	190.78	3.30e+04	838.35	1.40e+03	**1.29e−03**	1.38e+03	3.02e+04
f5		1.08e+04	2.21e+04	2.12e+03	6.35e+03	**269.03**	9.36e+03	3.44e+03	5.25e+03	*8.62e−09*
		1.47e+03	8.55e+03	385.96	1.55e+03	**157.47**	2.12e+03	777.42	2.83e+03	*8.53e−09*
f1	50	2.04e−13	7.56e+04	6.51e−05	27.52	1.10e−13	5.05e−03	**0**	2.26e−13	5.87e−09
		3.20e−14	5.05e+04	3.53e−04	2.80	1.44e−16	1.08e−03	**0**	1.42e−13	8.59e−10
f2		1.61e+04	1.10e+05	303.67	9.56e+03	4.17e+03	491.13	**0**	7.42e+04	7.86e−09
		4.05e+03	7.06e+04	256.31	3.80e+03	1.53e+03	172.20	**0**	5.27e+04	7.24e−10
f3		1.08e+07	1.80e+09	3.89e+06	1.36e+07	2.04e+07	1.05e+07	**2.59e+05**	1.49e+09	*6.14e−09*
		2.65e+06	1.40e+09	1.42e+06	7.15e+06	6.84e+06	3.73e+06	**7.85e+04**	5.24e+08	*6.86e−10*
f4		6.42e+04	1.87e+05	2.98e+03	3.01e+05	2.44e+04	2.16e+04	**152.51**	1.62e+05	4.68e+05
		1.24e+04	5.48e+04	1.23e+03	1.25e+05	1.06e+04	7.50e+03	**118.64**	9.19e+04	3.11e+05
f5		2.54e+04	3.19e+04	**3.33e+03**	1.94e+04	**3.43e+03**	2.06e+04	8.84e+03	1.10e+04	*2.85*
		2.54e+03	1.12e+04	**556.06**	3.50e+03	**646.04**	4.59e+03	1.59e+03	4.69e+03	*4.32*

is still considered a foraging algorithm as the movement of individuals is still influenced by the foraging behavior of bacteria but without communication between them.

For GA, the number of individuals was set to 40. We adopted a roulette-wheel selection process, a one-point crossover, and Gaussian mutation. The mutation probability was set to $\frac{1}{D}$, where D is the number of dimensions. The crossover and mutation operators were performed until producing an offspring that is equal to the current population in size, both were then merged and the best 40 individuals were chosen as the next population. The used crossover and mutation operators were chosen after experimenting with different combinations of the one-point, two-point, and arithmetic crossover operators as well as the uniform and Gaussian mutation operators.

For SPSO, the only parameter set is the swarm size and it was set to 40. The parameter values for w,c_1, and c_2 are already set in the code as 1.193, 1.193, and 0.721 respectively.

For ACO_R, we followed the same parameter settings in [36] having two ants $m = 2$, an archive size $k = 50$, $q = 10^{-4}$, and $\xi = 0.85$.

3.3. Experimental results and discussion

The results are provided in Table 1 for uni-modal functions, Tables 2 and 3 for multi-modal functions, and Tables 4 and 5 for the hybrid functions. The best results are highlighted in bold (to test for the significance of the results, we used the Mann–Whitney non-parametric statistical test, where the null hypothesis is rejected with a 95% confidence level). As IPOP-CMA-ES [3] is one of the most effective evolutionary algorithms up-to-date,[1] its results are added to the comparison. If the solution provided by IPOP-CMA-ES is as good as or better than the algorithms studied in this work, the result is shown in italic. In other words, a bold result means it is better than all the results provided by all the other algorithms but not necessarily better than IPOP-CMA-ES.

This section is organized as follows: Section 3.3.1 covers the results for the foraging algorithms. The performance of evolutionary algorithms is discussed in Section 3.3.2. The speed of convergence of all the algorithm is inspected in Section 3.3.3. Section 3.3.4 gives a brief comparison of all the algorithms against IPOP-CMA-ES. Section 3.3.5 highlights some of the issues regarding the degree of diversity maintained by some of the studied algorithms.

[1] IPOP-CMA-ES was the best submission to the CEC2005 special session on *Real – Parameter Optimization*: http://www.lri.fr/~hansen/cec2005.html.

Table 2
Average solutions provided by all the algorithms for the multi-modal CEC05 functions taken over 30 runs, 10 and 30 dimensions.

Benchmark function	Size	ABC	BA	HS_{Pop}	BFOA	DE	GA	SPSO	ACO_R	IPOP-CMA-ES
f6	10	1.46	3.78e+08	36.34	703.97	**3.76e−06**	31.56	39.89	19.84	*4.87e−09*
		1.63	5.11e+08	30.29	1.75e+03	**6.16e−06**	33.95	113.23	70.06	*1.66e−09*
f7		2.45e−01	1.72e+03	48.61	485.67	2.01e−01	6.93e−01	**4.36e−02**	6.14e−01	*3.31e−09*
		1.27e−01	659.32	55.78	156.15	1.83e−01	4.31e−01	**1.96e−02**	2.28e−01	*2.02e−09*
f9		**0**	54.42	**0**	15.62	**0**	5.61e−04	5.26	8.72	2.39e−01
		0	18.60	**0**	2.04	**0**	3.80e−04	2.74	4.82	4.34e−01
f10		28.59	79.73	20.62	25.16	10.80	21.99	**4.59**	27.81	*7.96e−02*
		8.29	21.52	3.46	3.49	9.27	12.45	**1.96**	7.00	*2.75e−01*
f11		5.38	9.15	8.42	9.65	9.00	6.67	**2.97**	8.53	*9.34e−01*
		6.04e−01	1.37	2.05	5.55e−01	8.10e−01	1.30	**1.52**	7.51e−01	*9.00e−01*
f12		**299.39**	3.18e+04	2.58e+03	696.90	1.08e+04	561.10	5.97e+03	2.61e+04	*29.30*
		177.41	1.21e+04	1.86e+03	189.29	7.69e+03	652.16	3.00e+03	8.56e+03	*142.00*
f13		**2.19e−01**	10.04	9.07e−01	5.06	1.43	3.41e−01	8.60e−01	1.87	6.96e−01
		8.95e−02	4.24	4.23e−01	6.88e−01	8.65e−01	1.33e−01	2.18e−01	6.14e−01	1.50e−01
f14		3.34	3.96	**2.33**	3.83	3.74	3.35	**2.47**	3.80	3.01
		2.12e−01	2.08e−01	**7.71e−01**	3.89e−01	6.04e−01	3.31e−01	**3.93e−01**	3.34e−01	3.49e−01
f6	30	**7.49**	1.24e+10	193.82	5.24e+03	59.01	129.89	141.65	537.65	*5.90e−09*
		10.99	1.37e+10	476.10	3.30e+03	45.27	59.75	226.24	2.66e+03	*1.61e−09*
f7		**1.23e−02**	3.37e+03	19.23	6.84e+03	5.46e−02	3.83e−01	**2.04e−02**	3.14e−01	*5.31e−09*
		5.04e−03	2.99e+03	21.78	441.70	5.37e−02	7.37e−02	**1.57e−02**	3.47e−01	*1.44e−09*
f9		**6.06e−14**	311.14	9.61	137.79	3.32e−02	1.52e−03	44.37	54.78	9.38e−01
		1.44e−14	81.88	14.03	11.67	1.82e−01	4.29e−04	10.15	40.84	1.18
f10		333.43	554.07	159.08	175.78	211.51	196.05	**46.13**	210.55	*1.65*
		68.97	77.66	8.29	13.81	40.38	40.39	**11.91**	17.93	*1.35*
f11		**27.94**	37.53	39.37	40.44	39.30	29.71	30.06	39.24	*5.38*
		1.62	5.01	1.60	1.10	1.22	2.90	2.85	1.24	*3.13*
f12		**8.55e+03**	1.06e+06	5.27e+05	4.03e+04	3.46e+05	1.30e+04	2.96e+05	1.02e+06	4.43e+04
		3.70e+03	1.39e+05	6.61e+04	6.16e+03	1.71e+05	6.80e+03	6.28e+04	1.13e+05	2.15e+05
f13		**9.44e−01**	101.62	11.98	27.36	5.92	1.21	4.37	14.32	2.49
		1.19e−01	85.84	1.19	2.15	3.17	2.61e−01	1.27	2.86	5.13e−01
f14		12.97	13.67	13.27	13.35	14.26	13.01	**12.44**	13.81	12.90
		2.16e−01	2.43e−01	1.90e−01	3.99e−01	1.33e−01	4.22e−01	**3.76e−01**	1.65e−01	4.19e−01

Table 3
Average solutions provided by all the algorithms for the multi-modal CEC05 functions taken over 30 runs, 50 dimensions.

Benchmark function	Size	ABC	BA	HS_{Pop}	BFOA	DE	GA	SPSO	ACO_R	IPOP-CMA-ES
f6	50	**6.16**	4.08e+10	367.58	1.57e+04	98.45	174.11	219.74	344.97	*7.13e−09*
		7.76	3.64e+10	1.27e+03	7.43e+03	57.68	61.46	243.62	979.21	*1.11e−09*
f7		**1.68e−04**	1.14e+04	29.59	1.12e+04	1.57e−01	3.94e−01	9.66e−03	3.05e−01	*7.22e−09*
		1.23e−04	3.47e+03	35.04	619.28	1.01e−01	8.37e−02	1.61e−02	3.54e−01	*1.03e−09*
f9		**1.14e−13**	637.11	87.81	378.26	1.99e−01	3.21e−03	119.56	189.86	1.39
		1.02e−28	170.82	50.38	33.60	4.82e−01	1.09e−03	25.62	105.92	1.11
f10		1.08e+03	1.12e+03	333.50	413.73	418.11	502.79	**130.47**	451.90	*1.72*
		102.68	201.78	9.91	30.93	21.90	103.01	**25.06**	37.26	*1.42*
f11		**56.24**	68.06	72.77	73.66	72.88	**56.76**	**57.28**	72.96	*11.7*
		2.23	7.59	1.08	1.26	1.52	**6.02**	**3.84**	1.49	*3.14*
f12		**3.52e+04**	5.09e+06	2.52e+06	1.78e+05	8.53e+05	**3.82e+04**	1.60e+06	5.49e+06	2.27e+05
		1.05e+04	6.72e+05	2.65e+05	2.77e+04	2.59e+05	**2.12e+04**	2.94e+05	4.00e+05	1.11e+06
f13		**1.63**	642.63	26.11	57.61	8.70	1.97	9.02	34.79	4.59
		1.97e−01	623.39	2.12	3.13	3.66	3.04e−01	3.51	4.26	5.15e−01
f14		22.68	23.38	23.06	23.13	24.15	22.63	**22.01**	23.78	22.90
		2.05e−01	2.25e−01	1.15e−01	3.54e−01	1.37e−01	4.14e−01	**4.33e−01**	1.73e−01	5.78e−01

3.3.1. Foraging algorithms

ABC emerges as the best performer on functions f9 and f15 as well as performing very well on f1. These functions are the only separable functions of the CEC05 library. This indicates that ABC is well suited for separable functions, specially multi-modal ones. The reason for this is that the update equation for ABC only modifies one problem variable at a time after which the new solution is evaluated. The same behavior is observed for a different testbed in [11]. A recent modification, although not tested in this work, to ABC was introduced in [2], where the number of modified problem variables could be controlled.

As for multi-modal functions, the results in Tables 2 and 3 show that ABC has good scalability. Although ABC was outperformed by DE and SPSO on functions f6, f7, and f11 for a dimensionality of 10, increasing the dimensionality to 30 and 50 had a much worse effect on the competing algorithms.

Table 4
Average solutions provided by all the algorithms for the hybrid CEC05 functions taken over 30 runs, 10 dimensions.

Benchmark function	Size	ABC	BA	HS_{Pop}	BFOA	DE	GA	SPSO	ACO_R	IPOP-CMA-ES
f15	10	**9.14e−02**	545.27	305.22	442.17	16.82	241.03	321.70	422.88	228.00
		4.07e−01	164.43	172.31	143.28	18.58	216.62	140.68	170.15	68.00
f16		145.20	308.63	138.75	165.04	109.77	144.63	**103.51**	220.89	*91.30*
		14.70	72.09	11.93	52.21	7.11	20.38	**8.77**	119.68	*3.40*
f17		152.61	350.22	149.79	227.25	143.85	157.84	**117.60**	237.97	*123.00*
		16.70	102.62	8.99	96.23	37.65	30.01	**11.53**	148.87	*20.90*
f18		554.08	1.10e+03	855.60	843.10	**407.51**	997.37	637.46	944.51	*332.00*
		86.66	88.65	150.13	109.10	**222.01**	108.79	265.94	80.72	*112.00*
f19		556.74	1.10e+03	841.99	721.08	**350.00**	984.77	686.87	929.59	*326.00*
		104.46	105.99	140.68	211.83	**152.56**	89.94	249.09	131.74	*99.30*
f20		560.35	1.11e+03	864.96	817.86	**465.01**	1.01e+03	618.64	969.72	*300.00*
		104.74	64.89	87.33	142.14	**261.61**	78.35	267.62	61.04	*0*
f21		**364.00**	1.30e+03	1.03e+03	665.17	500.00	1.01e+03	655.56	1.09e+03	500.00
		154.76	71.25	163.59	267.92	1.73e−13	255.89	250.30	164.61	3.48e−13
f22		**726.86**	1.04e+03	790.50	797.20	772.11	806.15	771.44	874.39	*729.00*
		169.78	55.48	40.68	89.72	17.47	117.24	23.26	52.05	*6.86*
f23		**488.67**	1.35e+03	1.08e+03	1.01e+03	559.47	995.16	798.85	1.09e+03	559.00
		62.92	47.12	183.86	214.49	1.16e−13	255.93	214.82	207.52	3.24e−11
f24		**200.00**	1.22e+03	**200.00**	470.05	**200.00**	277.44	273.33	540.96	*200.00*
		2.61e−03	281.43	**0**	271.07	**0**	131.83	161.11	265.55	*2.29e−06*
f25		**263.16**	1.39e+03	322.53	487.92	382.14	1.20e+03	426.65	535.39	347.00
		88.81	41.07	112.11	191.29	3.10	229.84	36.68	267.36	3.22

Table 5
Average solutions provided by all the algorithms for the hybrid CEC05 functions taken over 30 runs, 30 dimensions.

Benchmark function	Size	ABC	BA	HS_{Pop}	BFOA	DE	GA	SPSO	ACO_R	IPOP-CMA-ES
f15	30	**5.97e−01**	853.67	300.00	518.14	63.51	198.60	439.56	659.45	208.00
		1.74	192.91	26.26	94.82	50.84	206.10	73.23	142.94	27.50
f16		314.67	662.55	246.57	438.06	252.58	**142.38**	203.06	443.80	*35.00*
		49.35	145.46	110.89	112.67	45.17	**24.27**	175.03	162.85	*20.40*
f17		219.71	729.35	269.05	491.71	302.64	**143.67**	237.04	438.48	291.00
		32.98	133.18	119.06	157.76	54.32	**25.91**	153.46	122.61	193.00
f18		899.55	1.17e+03	910.11	915.26	903.74	864.64	916.25	**829.03**	904.00
		96.78	90.19	2.65	20.21	5.16e−01	135.45	5.70	**3.57**	2.99e−01
f19		915.46	1.18e+03	909.70	918.16	**903.90**	960.57	917.15	849.42	*904.00*
		22.24	72.97	3.42	23.70	**7.84e−01**	79.60	8.35	78.73	*2.48e−01*
f20		920.33	1.19e+03	909.38	919.17	903.75	959.96	916.10	**829.40**	889.00
		3.91	96.88	2.08	23.21	6.00e−01	137.06	5.79	**4.81**	45.5
f21		**488.36**	1.36e+03	500.00	671.37	519.39	909.72	674.45	869.97	500.00
		30.20	61.55	2.31e−013	209.06	106.23	288.22	294.25	5.58	1.13e−013
f22		1.07e+03	1.39e+03	891.20	1.12e+03	882.13	804.03	889.22	**552.79**	803.00
		36.03	95.52	14.29	57.93	11.10	519.97	15.47	**95.26**	18.60
f23		**531.97**	1.39e+03	555.00	873.54	552.92	994.47	710.92	904.27	*534.00*
		5.80	40.66	114.13	175.24	102.74	265.11	275.98	79.94	*2.24e−04*
f24		**200.00**	1.43e+03	**200.00**	1.16e+03	916.66	335.12	226.36	217.14	910.00
		0	31.63	**0**	250.37	194.91	159.36	144.38	3.28	148.00
f25		**203.19**	1.45e+03	220.92	1.17e+03	211.24	404.97	210.57	211.00	691.00
		6.93	17.84	14.86	113.78	9.65e−01	347.75	2.10e−01	6.26	9.21e−01

For the results of the hybrid functions shown in Tables 4 and 5, ABC managed to compete very well with the DE and SPSO evolutionary algorithms.

For functions f7 and f25 that have their optimum outside of the initial boundary, ABC has performed very well against another two algorithms that have the re-initializing component, which are BA and HS_{Pop}. The difference is that although ABC might produce new solutions in the initialization domain, these solutions can be brought back quickly to promising regions through the update equation as it involves cooperation with other bees.

On the other hand, both BA and HS_{Pop} employ a simple neighborhood search around the current solution (or another solution from memory as in HS_{Pop}). This operator might take a large number of steps in order to pull these solutions out of the initialization domain into good regions of the search space. Another reason is that both BA and HS_{Pop} apply this operator in every iteration while ABC applies it only to unimproved solutions.

The worst performing algorithms in this category are BA and BFOA as they are outperformed by either ABC or ACO_R in a large number of cases.

For BFOA, the reason for this is that some actions taken by this algorithm although biologically inspired might not be entirely suitable for function optimization. For example, the reproduction scheme employed by BFOA generates a population of S individuals from which $\frac{S}{2}$ are distinct. Employing a more effective scheme like what is used in GAs or DE is expected to improve the algorithm's performance. Another example is the approach used for calculating the health of the bacteria. Calculating the health based on the best fitness that each bacterium experienced during the chemotaxis steps might be better than summing over all the fitness values.

For BA, previously presented comparison against ABC provides the reason for this deteriorated performance.

Both BA and BFOA require a large number of parameters to be set. To the best of our knowledge, there is no clear study for either BFOA or BA on how to appropriately set their parameters, changing a single parameter can greatly affect the performance. Improving the performance of BFOA on uni-modal functions could be achieved by increasing the step size C or the number of steps the bacteria are allowed to swim in a single iteration Ns. This will enable the swarm to reach the global optimum faster. Improving the performance of BA could be achieved by manipulating the neighborhood size ngh.

For ACO_R, employing two ants only might not be sufficient to produce good results overall. Although in [36], the authors stated that the solutions in the archive constitute the population being updated from one iteration to the next, having more ants is as important. A good property for the tested ACO_R however is the ability to produce good results for functions with high conditioning values. Inspecting the results for function f22 (which is the same as f21 but with increased conditioning) shows that ACO_R produced an improved solution for f22 over the solution provided for f21.

Table 6
Performance rates and success rates of all the algorithms for the CEC05 uni-modal functions successfully solved.

	Size	Number of solved functions	Normalized performance rates			
			f1	f2	f4	f5
Best performance rates	10		2480	11360	15400	11880
ABC		1	4.12 100%	–	–	–
BA		0	–	–	–	–
HS_{Pop}		2	1.79 100%	–	–	1.01e+0 6.67%
BFOA		0	–	–	–	–
DE		4	8.79 100%	7.23 100%	6.08 100%	3.37 100%
GA		0	–	–	–	–
SPSO		4	2.36 100%	1.13 100%	1 100%	1 100%
ACO_R		3	1 100%	1 100%	1.54 100%	–
Best performance rates	30		12520	97200	–	–
ABC		1	2.42 100%	–	–	–
BA		0	–	–	–	–
HS_{Pop}		1	1 100%	–	–	–
BFOA		0	–	–	–	–
DE		1	32.07 73.3%	–	–	–
GA		0	–	–	–	–
SPSO		2	1.01 100%	1 100%	–	–
ACO_R		1	1.09 100%	–	–	–
Best performance rates	50		20320	280160	–	–
ABC		1	2.5 100%	–	–	–
BA		0	–	–	–	–
HS_{Pop}		1	1.20 93.33%	–	–	–
BFOA		0	–	–	–	–
DE		1	23.90 100%	–	–	–
GA		0	–	–	–	–
SPSO		2	1 100%	1 100%	–	–
ACO_R		1	1.88 100%	–	–	–

3.3.2. Evolutionary algorithms

The code for SPSO (version 2007) provides an option for using a rotated hypercube for the probability distribution used to generate random numbers in order to make the algorithm less sensitive to rotations. By setting this parameter in our experiments, SPSO shows its immunity against function rotation. By inspecting the results for all the algorithms on functions f9 (shifted Rastrigin) and f10 (shifted rotated Rastrigin), SPSO was the only algorithm not affected by this rotation. SPSO produced almost the same results for both functions across all dimensions while the performances of other algorithms have deteriorated considerably. These algorithms are rotationally variant as most calculations are done on a coordinate-by-coordinate basis. However, DE, for example, could be made rotationally invariant by eliminating the crossover operator as it relies on the coordinate system [4].

The results of the uni-modal functions in Table 1 show the superior performance of SPSO for these functions with the exception of f5 where DE produced the best results.

Out of all the studied algorithms, DE is the best performer in functions having their optimum near the bounds of the search space. This is the case for all types of functions, uni-modal (as in f5) or hybrid (as in f20) and for all dimensions. Although the reason behind this behavior may require more investigation, it could be due to the fact that the update equation of DE does not allow the population to converge prematurely.

For the GA, more advanced operators are needed to produce better results and compete with the studied algorithms. Nevertheless, although having a poor performance on the uni-modal functions, the basic studied approach showed better per-

Table 7
Performance rates and success rates of all the algorithms for the CEC05 multi-modal and hybrid functions successfully solved.

	Size	Number of solved functions	Normalized performance rates			
			f6	f7	f9	f15
Best performance rates	10		72160	–	11309	23097
ABC		2	–	–	1 100%	1 90%
BA		0	–	–	–	–
HS_{Pop}		1	–	–	3.26 100%	–
BFOA		0	–	–	–	–
DE		2	1 100%	–	1.36 100%	–
GA		1	–	–	7.36 90%	–
SPSO		0	–	–	–	–
ACO_R		1	1.55 66.67%	–	–	–
Best performance rates	30		1531920	58320	57769	292074
ABC		4	3.08 3.33%	21.23 20%	1 100%	1 80%
BA		0	–	–	–	–
HS_{Pop}		1	–	–	8.38 60%	–
BFOA		0	–	–	–	–
DE		1	–	154.25 3.33%	–	
GA		1	–	–	2.16 100%	–
SPSO		1	–	1 50%	–	
ACO_R		1	1 16.67%	–	–	–
Best performance rates	50		13569600	49560	107912	–
ABC		2	–	5.76 100%	1 100%	–
BA		0	–	–	–	–
HS_{Pop}		0	–	–	–	–
BFOA		0	–	–	–	–
DE		0	–	–	–	–
GA		1	–	–	2.63 100%	–
SPSO		1	–	1 100%	–	–
ACO_R		1	1 3.33%	–	–	–

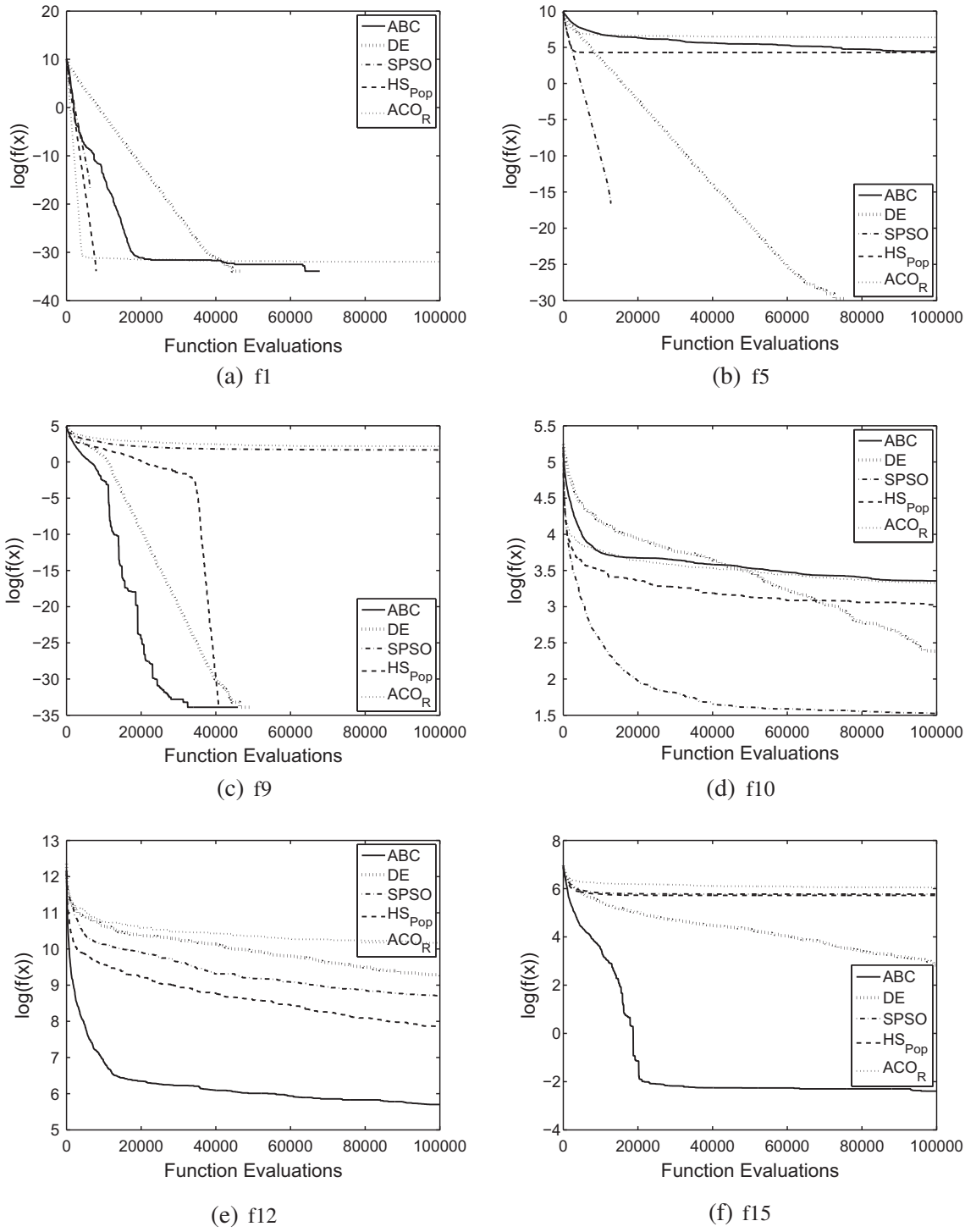

Fig. 1. Convergence behavior of all the algorithms for a sample of the CEC05 benchmark functions averaged over 30 runs, 10 dimensions.

formance than BA, BFOA, HS_{Pop}, and ACO_R in the multi-modal functions while competing with them on the hybrid functions. For more advanced GAs that have been applied to the CEC05 set, one can refer to [5,12].

3.3.3. Speed of convergence

Tables 6 and 7 show the success rates and performance rates for all the algorithms for the functions successfully solved. The tables show the best performance rates achieved by any of the algorithms accompanied by the normalized performance

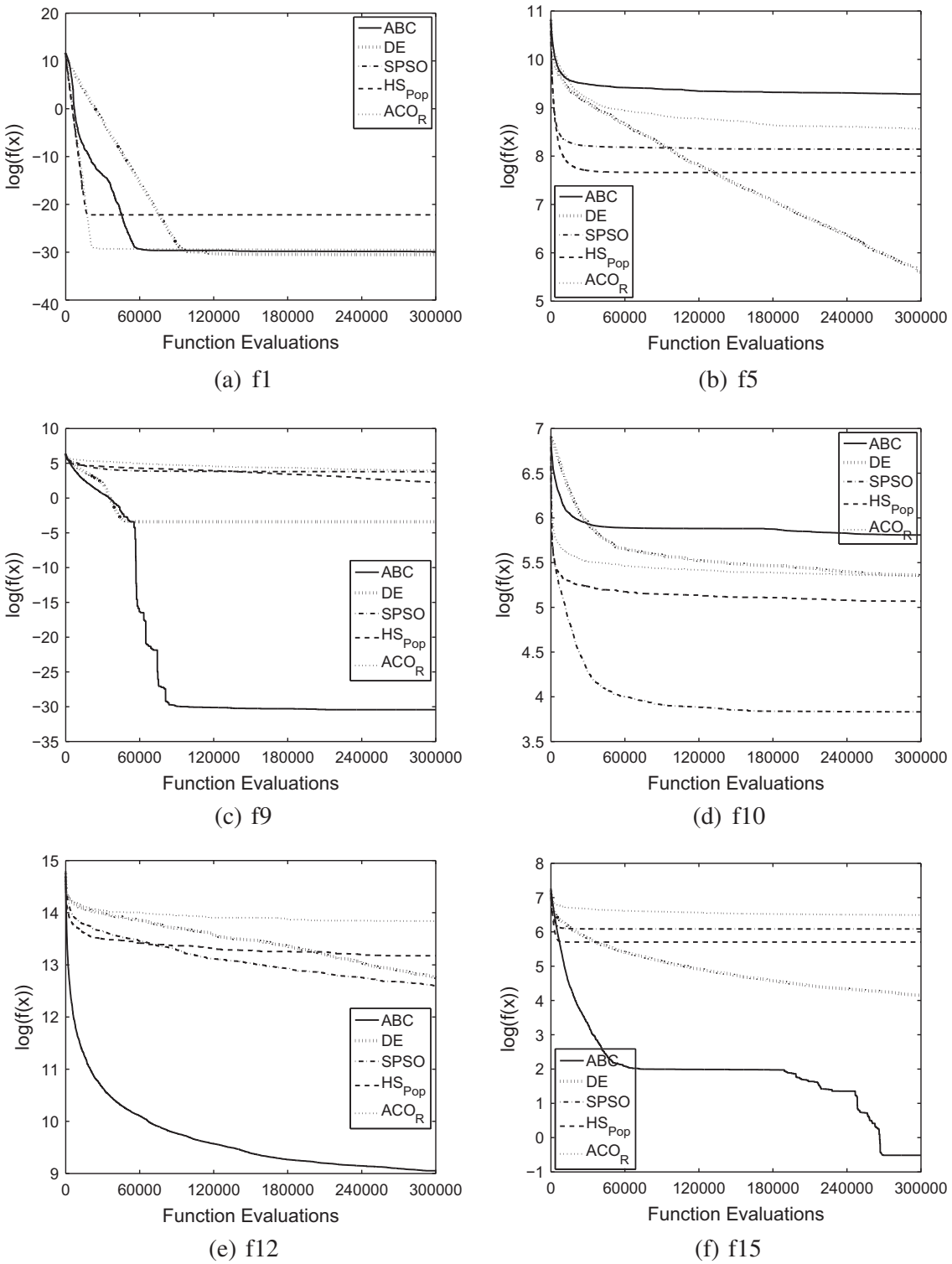

Fig. 2. Convergence behavior of all the algorithms for a sample of the CEC05 benchmark functions averaged over 30 runs, 30 dimensions.

rates for all the other algorithms (the best algorithm has a normalized entry of (1). An entry shown as "–" means that the corresponding algorithm did not reach the specified threshold in any of the runs.

Table 8
Comparison of the algorithms against IPOP-CMA-ES.

Algorithm	Uni-modal	Multi-modal	Hybrid	Total number
ABC	1	8	**11**	20
SPSO	**10**	3	1	14
Other algorithms	3	4	8	15
IPOP-CMA-ES	5	**12**	10	**27**

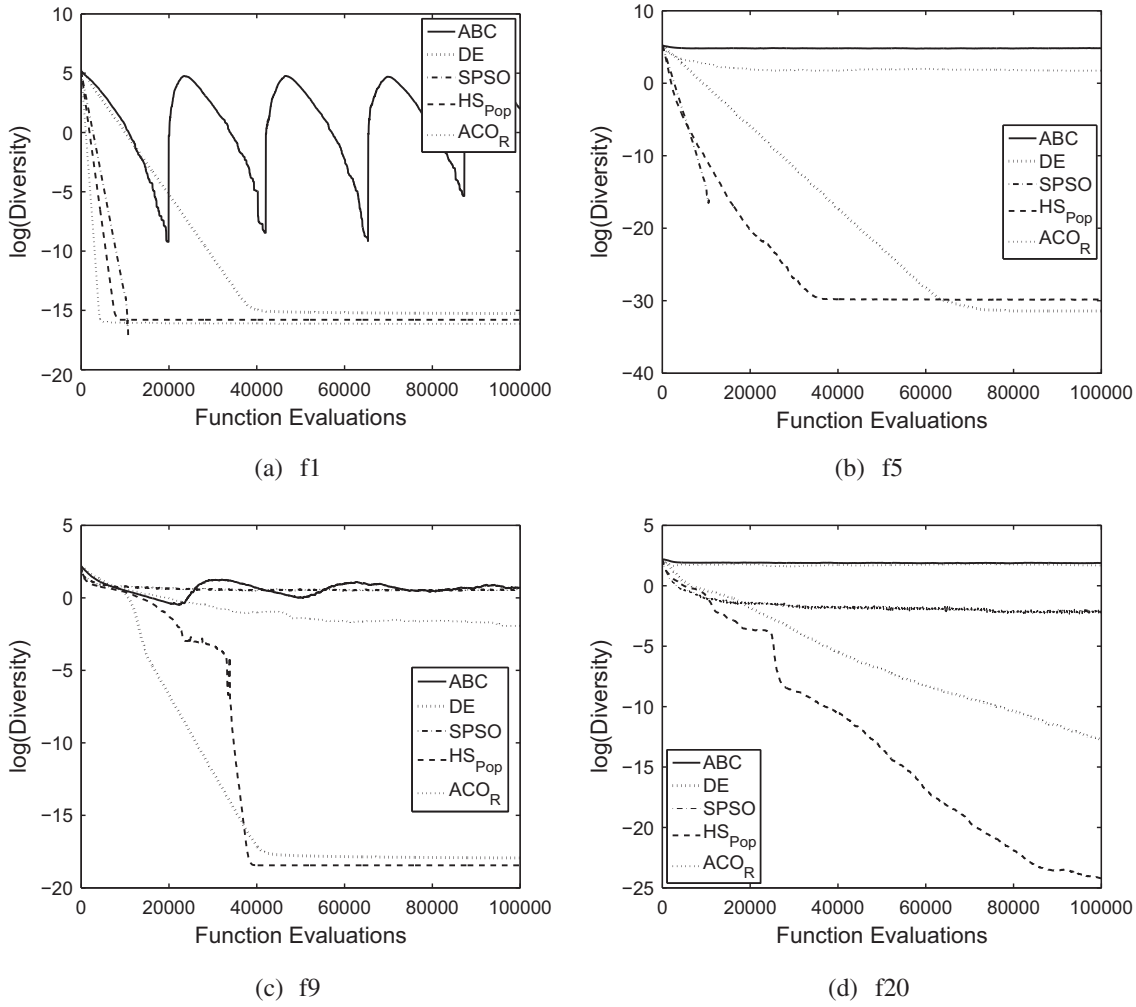

Fig. 3. Diversity of all the algorithms for a sample of the CEC05 benchmark functions averaged over 30 runs, 10 dimensions.

For all the successfully solved uni-modal problems, SPSO, ACO_R, and HS_{Pop} all show a fast speed of convergence (except for HS_{Pop} in f5). The results also illustrate the slow speed of convergence of DE. As stated in [34], DE needs much more function evaluations to produce good results.

The performance rates of ABC and SPSO show that for ABC to successfully solve a function, it requires on average double the number of function evaluations consumed by SPSO. This is due to the component generating random solutions in ABC, which induces diversity and delays convergence.

These results are also emphasized by the convergence plots shown in Figs. 1 and 2. The plots illustrate the fast speed of convergence exhibited by SPSO. In the function f1 successfully solved by many algorithms, the plots show that DE is the slowest algorithm while ABC has almost double the speed of convergence of SPSO. The plots for f1 and f5 illustrate that DE is still improving highlighting that it needs much more function evaluations to converge.

Although the diversity inducing component is available in both ABC and HS_{Pop}, ABC has a slower speed of convergence. The reason for this is that this component in ABC generates complete random solutions affecting the update equation for all

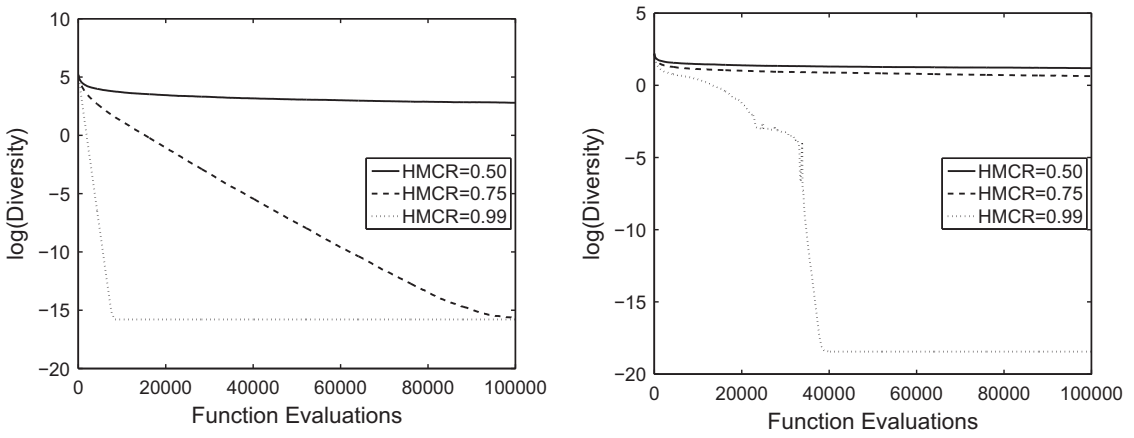

Fig. 4. Diversity of HS_{Pop} for functions f1 and f9 of the CEC05 benchmark functions averaged over 30 runs, 10 dimensions.

bees. However, in HS_{Pop}, it only induces randomly generated problem variables into a solution, which has a lesser effect than the ABC component. Another reason is that in HS_{Pop}, *HMCR* is set to 0.99, which makes it less frequent for this component to take effect.

The Performance rates and success rates for ABC confirm how ABC is suitable for separable functions as well as having the ability to scale up. ABC is able to reach the predefined threshold for f1 across all dimensions. ABC is also the only algorithm successful in solving f9 over all trials and across all dimensions. Moreover, ABC is the only algorithm able to successfully solve f15 for at least 80% of the runs.

For multi-modal functions, ABC still has a faster speed of convergence than DE but slower than SPSO (function f7 in dimensions 30 and 50). Although ACO_R seems to have produced good results for f6, the success rates show that it reached the predetermined threshold in just a few runs.

3.3.4. Comparison against IPOP-CMA-ES

Table 8 summarizes the previously presented results by showing the number of functions for which each algorithm has produced the best result. For uni-modal functions, SPSO is the best performer among all algorithms. For multi-modal functions, IPOP-CMA-ES is the best algorithm. For hybrid functions, both ABC and IPOP-CMA-ES have a comparable performance. Although IPOP-CMA-ES is the best performer overall, these results show that even the basic versions of the studied algorithms have a competitive performance with the state-of-the-art.

3.3.5. A note on diversity

To investigate the diversity characteristics for some of the studied algorithms, the diversity is plotted in Fig. 3. This diversity measure is calculated as:

$$Diversity = \frac{1}{S}\sqrt{\sum_{i=1}^{S}(\mathbf{x}_i - \bar{\mathbf{x}})^2}, \tag{21}$$

where $\mathbf{x}_i$ refers to the i^{th} individual in the population, S is the population size, and $\bar{\mathbf{x}}$ is the average position in the population.

A property that is observed for ABC in f1 and f9 is that the population diversity goes through cycles. The diversity drops reasonably fast but it increases again because of the diversity component that randomly initializes selected individuals in the specified domain. However, this cyclic behavior is not observed for non-separable functions. Overall, because of this component, the population in ABC always has a high degree of diversity across all the functions in comparison to the other algorithms.

SPSO uses an *lbest* topology with variable random neighborhood. This means that the movement of any particle is not influenced by the global best particle in the swarm but rather by the best particle in its neighborhood, a neighborhood that randomly changes in every iteration. This approach causes SPSO to have a high degree of diversity especially in multi-modal and hybrid functions as illustrated in the figure. Recent work in [13] proves that this model provides a wider search space for the particles than the classical *gbest* model.

An interesting observation that needs more investigation is that the diversity of both DE and HS_{Pop} continue to decrease through out the entire search process unlike other algorithms. However, in HS_{Pop}, this behavior could change as the diversity could be controlled through the *HMCR* parameter. As this parameter decreases, more random problem variables are

introduced into the current harmony. This behavior is illustrated in Fig. 4 as the population diversity of HS_{Pop} increases when *HMCR* is decreased.

4. Conclusions

This work provided a complete performance assessment of foraging optimization algorithms in comparison with evolutionary algorithms. The algorithms tested in this work are the artificial bee colony algorithm, the bees algorithm, ant colony optimization, the bacterial foraging optimization algorithm, harmony search, differential evolution, genetic algorithms, and standard particle swarm optimization. The work was conducted using the CEC05 benchmark functions for different problem sizes. The comparison was based on the solution reached, the success rate, and the performance rate.

Results obtained showed that SPSO is the best algorithm to be used with uni-modal functions having successfully solved the largest number of uni-modal functions while having a very fast speed of convergence. On average, ABC almost has double the speed of convergence of SPSO. For DE, more function evaluations are needed to provide good results.

For multi-modal and hybrid functions, ABC was the most successful algorithm. ABC was proven to be extremely successful when handling separable functions. On the other hand, SPSO was the most robust algorithm when faced with function rotations.

For BFOA, some techniques were proposed for improving its performance including adopting a different scheme for reproduction and/or different approach for the health calculation step.

The results also showed that the studied algorithms, although in their basic versions, have a very good performance in comparison with IPOP-CMA-ES. SPSO (ABC) is better than (as good as) IPOP-CMA-ES in uni-modal (hybrid) functions. This is despite the fact that IPOP-CMA-ES is equipped with a restart mechanism as well as an increasing population size between different restarts.

Through this study, different useful components of the tested algorithms were identified. First, to have a good population-based optimization algorithm, we believe that the update equation should exhibit some form of cooperation between the different individuals. Second, a good component is the ability to continuously generate new random solutions without compromising the quality of the search. Third, having an update equation that is not heavily steered by the best individual in the population prevents the population from quickly converging to inferior solutions.

Several directions exist for future work: first, we will focus on improving the performance of BFOA based on suggestions presented in this work. Second, we will study the effect of increasing the number of ants on the performance of ACO_R. Third, we will investigate the development of a hybrid ABC-SPSO algorithm that could exhibit the good characteristics highlighted in this study for both algorithms.

References

[1] B. Akay, D. Karaboga, Parameter tuning for the artificial bee colony algorithm, in: Proceedings of 1st International Conference on Computational Collective Intelligence, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 608–619.
[2] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Information Sciences (2010), doi:10.1016/j.ins.2010.07.015.
[3] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, in: Proceedings of IEEE Congress on Evolutionary Computation, vol. 2, 2005, pp. 1769–1776.
[4] A. Auger, N. Hansen, J.M. Perez Zerpa, R. Ros, M. Schoenauer, Experimental comparisons of derivative free optimization algorithms, Technical report, inria-00397334, 2010, France.
[5] P.J. Ballester, J. Stephenson, J.N. Carter, K. Gallagher, Real-parameter optimization performance study on the CEC-2005 benchmark with SPC-PNX, Proceedings of IEEE Congress on Evolutionary Computation, vol. 3019, Springer, Berlin, 2005, pp. 544–551.
[6] S. Bitam, M. Batouche, E. Talbi, A survey on bee colony algorithms, in: Proceedings of 24th IEEE/ACM International Parallel and Distributed Processing Symposium, IPDPS, 2010, pp. 1–8.
[7] C. Blum, A. Roli, Metaheuristics in combinatorial optimization: overview and conceptual comparison, ACM Computing Surveys 35 (3) (2003) 268–308.
[8] P. Chakraborty, G.G. Roy, S. Das, D. Jain, An improved harmony search algorithm with differential mutation operator, Fundamenta Informaticae 95 (2009) 1–26.
[9] S. Dasgupta, S. Das, A. Abrahamn, A. Biswas, Adaptive computational chemotaxis in bacterial foraging optimization: an analysis, IEEE Transactions on Evolutionary Computation 13 (4) (2009) 919–941.
[10] M. Dorigo, Optimization, Learning and Natural Algorithms (in Italian), Ph.D. Thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy, 1992.
[11] M. El-Abd, Black-box optimization benchmarking for noiseless function testbed using artificial bee colony algorithm, in: GECCO (Companion) in Black-Box Optimization Benchmarking Workshop, 2010, pp. 1719–1724.
[12] C. García-Martínez, M. Lozano, Hybrid real-coded genetic algorithms with female and male differentiation, Proceedings of IEEE Congress on Evolutionary Computation, vol. 3019, Springer, Berlin, 2005, pp. 544–551.
[13] S. Ghosh, S. Das, D. Kundu, K. Suresh, A. Abraham, Inter-particle communication and search-dynamics of lbest particle swarm optimizers: an analysis, Information Sciences 181 (1) (2011) 156–168.
[14] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical report, TR06, Engineering Faculty, Computer Engineering Department, Erciyes University, 2005.
[15] D. Karaboga, Artificial bee colony code. <http://mf.erciyes.edu.tr/abc/software.htm>, 2008.
[16] D. Karaboga, B. Akay, Artificial bee colony (ABC), harmony search and bees algorithms on numerical optimization, in: Proceedings of Innovative Production Machines and Systems Virtual Conference, IPROMS, 2009a.
[17] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Applied Mathematics and Computation 214 (2009) 108–132.
[18] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artificial Intelligence Review 31 (2009) 61–85.
[19] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (3) (2007) 459–471.
[20] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (1) (2008) 687–697.

[21] J. Kennedy, R. Mendes, Population structure and particle swarm performance, Proceedings of IEEE Congress on Evolutionary Computation, vol. 2, IEEE Computer Society, Washington, DC, 2002, pp. 1671–1676.
[22] P. Larrañaga, J.A. Lozano, Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation, Genetic Algorithms and Evolutionary Computation, vol. 2, Springer, Berlin, 2001.
[23] K.S. Lee, Z.W. Geem, A new meta-heuristic algorithm for continuous engineering optimization: harmony search theory and practice, Computer Methods in Applied Mechanics Engineering 194 (2005) 3902–3933.
[24] M. Mahdavi, M. Fesanghary, E. Damangir, An improved harmony search algorithm for solving optimization problems, Applied Mathematics and Computation 188 (2007) 1567–1579.
[25] A. Mukhopadhyay, A. Roy, S. Das, S. Das, A. Abraham, Population-variance and explorative power of harmony search: an analysis, in: Second National Conference on Mathematical Techniques: Emerging Paradigms for Electronics and IT Industries (MATEIT 2008), New Delhi, India, 2008.
[26] M.G.H. Omran, M. Mahdavi, Global-best harmony search, Applied Mathematics and Computation 198 (2008) 643–656.
[27] Particle Swarm Central, Standard pso 2007 code. <http://www.particleswarm.info>, 2007.
[28] K.M. Passino, Biomimicry of bacterial foraging for distributed optimization and control, IEEE Control Systems Magazine 5 (3) (2002) 52–67.
[29] K.M. Passino, Bacterial foraging optimization algorithm code. <http://www.ece.osu.edu/passino/ICbook/ic_code.html>, 2005.
[30] D.T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, M. Zaidi, The bees algorithm-a novel tool for complex optimisation problems, in: Proceedings of Innovative Produxction Machines and Systems Virtual Conference, IPROMS, 2006, pp. 451–461.
[31] K.V. Price, R. Storn, Differential evolution, Dr. Dobb's Journal (264) (1997) 18–24.
[32] K.V. Price, R. Storn, J. Lampinen, Differential Evolution – A Practical Approach to Global Optimization, Springer, Heidelberg, 2005.
[33] E. Rashedi, H. Nezamabadi-pour, S. Sarayzdi, GSA: a gravitational search algorithm, Information Sciences (179) (2009) 2232–2248.
[34] J. Ronkkonen, S. Kukkonen, K.V. Price, Real-parameter optimization with differential evolution, Proceedings of IEEE Congress on Evolutionary Computation, vol. 1, IEEE Computer Society, Washington, DC, 2005, pp. 506–513.
[35] S. Scholz, The bees algorithm code. <http://www.bees-algorithm.com/modules/3/1.php>, 2007.
[36] K. Socha, M. Dorigo, Ant colony optimization for continuous domains, European Journal of Operational Research 185 (3) (2008) 1155–1173.
[37] R. Storn, K. Price, Differential evolution code. <http://www.icsi.berkeley.edu/storn/code.html>, 2005.
[38] R. Storn, K.V. Price, Differential evolution – a fast and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization (11) (1997) 341–359.
[39] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, CEC05 benchmark functions. <http://staffx.webstore.ntu.edu.sg/MySite/Public.aspx?accountname=epnsugan>, 2005.
[40] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, Technical report, 2005005, ITT Kanpur, India, 2005.
[41] X. Yang, Harmony search as a metaheuristic algorithm, in: Z.W. Geem (Ed.), Music-Inspired Harmony Search Algorithm, Studies in Computational Intelligence, vol. 191, Springer-Verlag, Berlin Heidelberg, 2009, pp. 1–14.
[42] X.S. Yang, Nature-Inspired Metaheuristic Algorithms, Genetic Algorithms and Evolutionary Computation, Luniver Press, 2008.