Differential Evolution with Fitness Diversity Self-adaptation

Ville Tirronen and Ferrante Neri

Abstract. This chapter proposes the integration of fitness diversity adaptation techniques within the parameter setting of Differential Evolution (DE). The scale factor and crossover rate are encoded within each genotype and self-adaptively updated during the evolution by means of a probabilistic criterion which takes into account the diversity properties of the entire population. The population size is also adaptively controlled by means of a novel technique based on a measurement of the fitness diversity. An extensive experimental setup has been implemented by including multivariate problems and hard to solve fitness landscapes. A comparison of the performance has been conducted by considering both standard DE and modern DE based algorithms, recently proposed in the literature. Available numerical results show that the proposed approach seems to be very promising for some fitness landscapes and still competitive with modern algorithms in other cases. In most cases analyzed the proposed self-adaptation is beneficial in terms of algorithmic performance and can be considered a useful tool for enhancing the performance of a DE scheme.

1 Introduction

Differential Evolution (DE, see [33], [29], and [6]) is a reliable and versatile function optimizer. DE, like most popular Evolutionary Algorithms (EAs), is a population based tool. DE, unlike other EAs, generates offspring by perturbing the solutions with a scaled difference of two randomly selected

Ville Tirronen · Ferrante Neri
Department of Mathematical Information Technology, Agora, University of Jyväskylä, P.O. Box 35 (Agora), FI-40014 University of Jyväskylä, Finland
e-mail: aleator@cc.jyu.fi,neferran@cc.jyu.fi

R. Chiong (Ed.): Nature-Inspired Algorithms for Optimisation, SCI 193, pp. 199–234.
springerlink.com

population vectors, instead of recombining the solutions by means of a probability function. In addition, DE employs a steady state logic which allows replacement of an individual only if the offspring outperforms its corresponding parent. Due to its algorithmic structure, over the optimization process DE generates a super-fit individual which leads the search until an individual with better performance is generated. Therefore, as highlighted in [16], a DE population can be subject to stagnation in such cases where no offspring individuals outperform the corresponding parents for a large number of generations. In order to avoid this undesired effect a proper parameter setting (two parameters in particular) is crucial.

Some empirical studies carried out in the literature, for example [28] and [16], give some hints as to how to perform such settings. A more advanced criterion based on the fitness values is given in [1]. Paper [36] proposed a dynamic population sizing strategy based on self-adaptation, and [19] proposed the employment of a fuzzy controller in order to perform setting of the parameters. In order to avoid execution of the parameter setting, in [30] an adaptive system based on the combined use of two learning strategies has been proposed. Paper [2] proposed a simple probabilistic scheme with a self-adaptive logic for updating parameter values during the evolution. Although the algorithm updates the parameters only by periodically refreshing them by means of random values, the self-adaptive logic carried out seems very robust and shows good performance under various fitness landscapes. This algorithmic philosophy has also been extended in the case of constrained optimization [3] and multi-objective problems [43]. Some hybrid approaches (also known as Memetic Algorithms) consisting of a DE framework and local search components have been proposed in the literature in order to avoid stagnation problems and, more generally, enhance the performance of the DE. Paper [35] proposed a hybrid algorithm based on a combination of DE and an estimation of distribution algorithm. This technique uses a probability model to detect promising areas and then focuses the search process on those regions. Recently, [26] proposed a Memetic Algorithm which integrates the local search hill-climber within variation operators of the DE; [31] proposed a heuristic technique, namely, opposition-based learning (OBL) for population initialization and also for generation jumping. In [42] a complex self-adaptation is proposed in order to significantly enhance the performance of a DE framework. Papers [7] and [8] propose the integration of a neighborhood logic within a DE scheme and the employment of multiple scale factors in the mutation operator. Papers [37] and [38] proposed a Memetic Differential Evolution (MDE) composed of a DE framework and two local search algorithms integrated within the framework and coordinated by a fitness diversity logic.

Fitness diversity adaptation has recently been applied with success in the context of Memetic Algorithms for performing coordination of local search algorithms and parameter setting. Several control parameters have been

designed on the basis of the evolutionary framework and optimization problems under analysis (see [4], [23], [21], [24], and [22]). A measurement of the fitness diversity as the difference in performance between the fittest individual and other members of the population has been recently introduced for a MDE scheme [5].

This chapter aims to employ the fitness diversity logic and integrate it within each genotype of a DE population in order to enhance the algorithmic performance. The algorithm proposed in this chapter is composed of a DE structure employing a self-adaptation similar to the one proposed in [2], with a modified probabilistic criterion which is based on a novel measurement of the fitness diversity. In addition, the proposed algorithm contains an adaptive population size determined by variations in the fitness diversity.

Section 2 presents the state-of-the-art regarding DE and five recently published DE based algorithms. Section 3 describes the concept of fitness diversity and the reason for its employment in algorithmic adaptation. Section 4 gives a description of the proposed algorithm, namely Fitness Diversity Self-Adaptive Differential Evolution. Section 5 shows the numerical results. Finally, Section 6 gives the conclusion to this chapter.

2 Background: Differential Evolution Based Algorithms

This section describes the DE and five DE based algorithms recently proposed. In order to clarify the notation used throughout this chapter we refer to the minimization problem of an objective function $f(x)$, where x is a vector of n design variables in a decision space D.

2.1 Differential Evolution

According to its original definition given in [33], the DE consists of the following steps. An initial sampling of S_{pop} individuals is performed pseudo-randomly with a uniform distribution function within the decision space D. At each generation, for each individual x_i of the S_{pop}, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. According to DE logic, a provisional offspring x'_{off} is generated by mutation as:

$$x'_{off} = x_t + F(x_r - x_s) \tag{1}$$

where $F \in [0, 1+[$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. The mutation scheme shown in eq. (1) is also known as DE/rand/1. Other variants of the mutation rule have been subsequently proposed in literature, see [30]:

- DE/best/1: $x'_{off} = x_{best} + F(x_s - x_t)$
- DE/cur-to-best/1: $x'_{off} = x_i + F(x_{best} - x_i) + F(x_s - x_t)$

- DE/best/2: $x'_{off} = x_{best} + F\,(x_s - x_t) + F\,(x_u - x_v)$
- DE/rand/2: $x'_{off} = x_r + F\,(x_s - x_t) + F\,(x_u - x_v)$

where x_{best} is the solution with the best performance among the individuals of the population, x_u and x_v are two additional pseudo-randomly selected individuals.

Then, to increase exploration, each gene of the new individual x'_{off} is switched with the corresponding gene of x_i with a uniform probability and the final offspring x_{off} is generated:

$$x_{off,j} = \begin{cases} x_{i,j} & \text{if} \quad rand\,(0,1) < CR \\ x'_{off,j} & otherwise \end{cases} \tag{2}$$

where $rand\,(0,1)$ is a random number between 0 and 1; j is the index of the gene under examination.

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) < f(x_i)$; otherwise no replacement occurs. For sake of clarity, the pseudo-code highlighting the working principles of the DE is shown in Fig. 1.

```
generate S_pop individuals of the initial population pseudo-randomly;
while budget condition
  for i = 1 : S_pop
    compute f(x_i);
  end-for
  for i = 1 : S_pop
    **mutation**
    select three individuals x_r, x_s, and x_t;
    compute x'_off = x_t + F(x_r − x_s);
    **crossover**
    x_off = x'_off;
    for j = 1 : n
      generate rand(0,1);
      if rand(0,1) < CR
        x_off,j = x_i,j;
      end-if
    end-for
    **selection**
    if f(x_off) < f(x_i)
      x_i = x_off;
    end-if
  end-for
end-while
```

Fig. 1 DE pseudocode

2.2 Parameter Setting in Differential Evolution

As highlighted in [16], due to its inner structure, the DE is subject to stagnation problems. Stagnation is that undesired effect which occurs when a population based algorithm does not converge to a solution (even suboptimal) and the population diversity is still high. In the case of the DE, stagnation occurs when the algorithm does not manage to improve upon any solution of its population for a prolonged amount of generations.

In order to avoid this undesired effect, the moving operators of the DE must be properly set. In other words, for successful functioning of the DE, a proper setting of the population size and parameters F and CR (see equations (1)and (2)) must be performed. The population size, analogous to the other Evolutionary Algorithms (EAs), if too small could cause premature convergence and if too large could cause stagnation (see [11]). A good value can be found by considering the dimensionality of the problem similar to what is commonly performed for the other EAs. A guideline is given in [34] where a setting of S_{pop} equal to ten times the dimensionality of the problem is proposed.

On the other hand, the setting of F and CR is neither an intuitive nor a straightforward task but is unfortunately crucial for guaranteeing the algorithmic functioning. Several studies have thus been proposed in literature. The study reported in [16] arrives at the conclusion, after an empirical analysis, that usage of $F = 1$ is not recommended, since according to a conjecture of the authors it leads to a significant decrease in explorative power. Analogously, the setting $CR = 1$ is also discouraged since it would dramatically decrease the amount of possible offspring solutions. In [34] and [17] the settings $F \in [0.5, 1]$ and $CR \in [0.8, 1]$ are recommended. In [17] the setting $F = CR = 0.9$ is chosen on the basis of discussion in [28]. The empirical analysis reported in [44] shows that in many cases the setting of $F \geq 0.6$ and $CR \geq 0.6$ leads to results having better performance.

Several studies, e.g., [13] and [18], highlight that an efficient parameter setting is very prone to problems (e.g., $F = 0.2$ could be a very efficient setting for a certain fitness landscape and completely inadequate for another problem). This result can be seen as a confirmation of the validity of the No Free Lunch Theorem [40] with reference to the DE schemes. In [1], a modified version of DE has been proposed. A system with two evolving populations has been proposed. The crossover rate CR has been set equal to 0.5 after an empirical study. Unlike CR, the value of F is adaptively updated at each generation by means of the following scheme:

$$F = \begin{cases} \max\left\{l_{\min}, 1 - \left|\frac{f_{\max}}{f_{\min}}\right|\right\} & \text{if } \left|\frac{f_{\max}}{f_{\min}}\right| < 1 \\ \max\left\{l_{\min}, 1 - \left|\frac{f_{\min}}{f_{\max}}\right|\right\} & \text{otherwise} \end{cases} \tag{3}$$

where $l_m = 0.4$ is the lower bound of F, f_{min} and f_{max} are the minimum and maximum fitness values over the individuals of the populations.

2.3 Self-adapting Control Parameters in Differential Evolution

In order to avoid the manual parameter setting of F and CR a simple and effective strategy has been proposed in [2]. This strategy is named Self-Adapting Control Parameters in Differential Evolution. The DE algorithm employing this strategy here is called Self-Adaptive Control Parameter Differential Evolution (SACPDE) and consists of the following.

With reference to Fig. 1, when the initial population is generated, two extra values between 0 and 1 are also generated per each individual. These values represent F and CR related to the individual under analysis. Each individual is thus composed (in a self-adaptive logic) of its genotype and its control parameters:

$$x_i = \langle x_{i,1}, x_{i,2}, ..., x_{i,j}, ...x_{i,n}, F_i, CR_i \rangle .$$

In accordance with a self-adaptive logic, see e.g., [32], the variation operations are preceded by the parameter update. More specifically when, at each generation, the i^{th} individual x_i is taken into account and three other individuals are extracted pseudo-randomly, its parameters F_i and CR_i are updated according to the following scheme:

$$F_i = \begin{cases} F_l + F_u rand_1, \text{ if } rand_2 < \tau_1 \\ F_i, \qquad\qquad \text{otherwise} \end{cases} \tag{4}$$

$$CR_i = \begin{cases} rand_3, \text{ if } rand_4 < \tau_2 \\ CR_i, \qquad \text{otherwise} \end{cases} \tag{5}$$

where $rand_j$, $j \in \{1, 2, 3, 4\}$, are uniform pseudo-random values between 0 and 1; τ_1 and τ_2 are constant values which represent the probabilities that parameters are updated, F_l and F_u are constant values which represent the minimum value that F could take and the maximum variable contribution to F, respectively. The newly calculated values of F_i and CR_i are then used for generating the offspring. The variation operators and selection scheme are identical to that of a standard DE (see section 2.1).

For sake of clarity, the pseudo-code highlighting the working principle of the SACPDE is given in Fig. 2.

2.4 Differential Evolution with Adaptive Crossover Local Search

In order to enhance performance of the DE, in [26] a memetic approach, called Differential Evolution with Adaptive Hill Climbing Simplex Crossover (DEahcSPX), has been proposed. The main idea is that a proper balance of the exploration abilities of the DE and the exploitation abilities of a Local

Searcher (LS) can lead to an algorithm with high performance. The proposed algorithm hybridizes the DE described in section 2.1 as an evolutionary framework and a LS is deterministically applied to the individual of the DE population with the best performance (in terms of fitness value).

The LS proposed for this hybridization is Simplex Crossover (SPX) [39]. More specifically, at each generation, that individual having the best fitness value, indicated here with x_b, is extracted and the LS described in Fig. 3 is applied. If the SPX succeeds in improving upon the starting solution, a replacement occurs according to a meta-Lamarckian logic [27].

It should be remarked that ϵ in Fig. 3 is a control parameter of the SPX which has been set equal to 1 in [26]. Finally, the DE framework employed is the standard DE described in Fig. 1.

generate S_{pop} individuals of the initial population with related parameters pseudo-randomly;
while budget condition
  for $i = 1 : S_{pop}$
    compute $f(x_i)$;
  end-for
  for $i = 1 : S_{pop}$
    F_i update
    generate $rand_1$ and $rand_2$;
    $F_i = \begin{cases} F_l + F_u rand_1, \text{ if } rand_2 < \tau_1 \\ F_i, \qquad\qquad \text{otherwise} \end{cases}$;
    mutation
    select three individuals x_r, x_s, and x_t;
    compute $x'_{off} = x_t + F_i(x_r - x_s)$;
    CR_i update
    generate $rand_3$ and $rand_4$;
    $CR_i = \begin{cases} rand_3, \text{ if } rand_4 < \tau_2 \\ CR_i, \qquad \text{otherwise} \end{cases}$
    crossover
    $x_{off} = x'_{off}$;
    for $j = 1 : n$
      generate $rand(0, 1)$;
      if $rand(0, 1) < CR_i$
        $x_{off,j} = x_{i,j}$;
      end-if
    end-for
    selection
    if $f(x_{off}) < f(x_i)$
      $x_i = x_{off}$;
    end-if
  end-for
end-while

Fig. 2 SACPDE pseudocode

while budget condition OR $f(C) \geq f(x_b)$
select pseudo-randomly $n_p - 1$ individuals from the DE population
compute the center of mass (including x_b):
$O = \frac{1}{n_p} \sum_{i=1}^{n_p} x_i$;
for $i = 1 : n_p - 1$
$r_i = rand(0,1)^{\frac{1}{i+1}}$;
end-for
for $i = 1 : n_p$
$y_i = O + \epsilon(x_i - O)$;
end-for
$C_1 = 0$;
for $i = 2 : n_p$
$C_i = r_{i-1}\,(y_{i-1} - y_i + C_{i-1})$;
end-for
$C = C_{n_p} + y_{n_p}$;
end-while

Fig. 3 SPX pseudocode

2.5 Opposition Based Differential Evolution

The Opposition Based Differential Evolution (OBDE), proposed in [31], employs logic of the opposition points in order to enhance exploration properties of the DE and test a wide portion of the decision space.

For a given point $x_i = \langle x_{i,1}, x_{i,2}, ..., x_{i,j}, ..., x_{i,n} \rangle$ belonging to a set $D = [a_1, b_1] \times [a_2, b_2] \times ... \times [a_j, b_j] \times ... \times [a_n, b_n]$ its opposition point is defined as: $\tilde{x}_i = \langle a_1 + b_1 - x_{i,1}, a_2 + b_2 - x_{i,2}, ..., a_j + b_j - x_{i,j}, ..., a_n + b_n - x_{i,n} \rangle$. The OBDE consists of a DE framework and two opposition based components: the first after the initial sampling and the second after the survivor selection scheme. While the first opposition based component is always applied after initialization, the second is activated by means of the probability j_r (jump rate). These opposition based components process a set of candidate solutions and generate their opposition points. They then merge the two sets of points (original and opposition) and select those points which have the best performance (as many as there are candidate solutions in the original set).

More specifically, when the initial sampling is pseudo-randomly performed, opposition points of the initial population are calculated and then half of these points (having the best fitness values) are selected to begin the optimization process. Analogously, at the end of each DE generation, when the population has been selected for the subsequent generation, the opposition based component is applied.

For sake of clarity the pseudo-code describing the functioning of the OBDE is shown in Fig. 4.

```
generate S_pop individuals of the initial population pseudo-randomly;
while budget condition
  for i = 1 : S_pop
    compute f(x_i);
  end-for
  **opposition based component**
  for i = 1 : S_pop
    for j = 1 : n
      compute x̃_{i,j} = a_j + b_j − x_{i,j};
    end-for
    compute f(x̃_i);
  end-for
  merge original population and opposition based points;
  select the S_pop solutions which have the best performance;
  for i = 1 : S_pop
    **mutation**
    select three individuals x_r, x_s, and x_t;
    compute x'_{off} = x_t + F(x_r − x_s);
    **crossover**
    x_{off} = x'_{off};
    for j = 1 : n
      generate rand(0, 1);
      if rand(0, 1) < CR
        x_{off,j} = x_{i,j};
      end-if
    end-for
    **selection**
    if f(x_{off}) < f(x_i)
      x_i = x_{off};
    end-if
  end-for
  generate rand(0, 1);
  if rand(0, 1) < j_r
    **opposition based component**
    for i = 1 : S_pop
      for j = 1 : n
        compute x̃_{i,j} = a_j + b_j − x_{i,j};
      end-for
      compute f(x̃_i);
    end-for
    merge original population and opposition based points;
    select the S_pop solutions which have the best performance;
  end-if
end-while
```

Fig. 4 OBDE pseudocode

2.6 Differential Evolution with Global and Local Neighborhoods

The Differential Evolution with Global and Local Neighborhoods (DEGL) modifies the mutation operation in the DE, explained in Subsection 2.1, by defining a neighborhood as a portion of the population identified by a radius k, see [7] and [8]. More specifically, individuals of the population are pseudo-randomly sorted and each individual is characterized by a position index i. The neighborhood of the i^{th} individual x_i is given by those individuals $x_{i-k}, ..., x_i, ..., x_{i+k}$.

The concept of neighborhood is used during the mutation operation since the provisional offspring x'_{off} is acquired through the combination of two contributions, the first contribution is given by the neighborhood individuals and the second by the whole population. Thus, in order to perform the mutation, for an individual x_i, the local contribution is calculated as:

$$L_i = x_i + \alpha\,(x_{n-best} - x_i) + \beta\,(x_p - x_q) \tag{6}$$

where x_{best} is the individual having best performance in the neighborhood, x_p and x_q and two individuals pseudo-randomly selected from the neighborhood. Values α and β are two constant which have a similar role to that of the scale factor F, see eq. (1). The global contribution is given by:

$$G_i = x_i + \alpha\,(x_{p-best} - x_i) + \beta\,(x_r - x_s) \tag{7}$$

where x_{p_best} is that individual with the best performance out of the entire population, x_r and x_s are two individuals pseudo-randomly selected from the population. The two contributions are then combined by means of:

$$x'_{off} = wG_i + (1 - w)\,L_i \tag{8}$$

where w_i is a weight factor to be set between 0 and 1.

Regarding the parameter setting, as suggested in [7], it has been set $\alpha = \beta$ equal to constant value. Also the neighborhood radius k has been set as a constant. On the contrary, the weight factor w is updated according to:

$$w = w_{\min} + (w_{\max} - w_{\min})\,\frac{g}{g_{\max}} \tag{9}$$

where $w_{\min}$ and $w_{\max}$ are the lower and upper bounds of the weight factor, respectively. The indexes g and $g_{\max}$ denote the current generation index and the maximum amount of generations, respectively. In other words, at the beginning of the optimization process ($g = 0$) the weight factor is set equal to w_{min} and subsequently linearly varies over time. At the end of the optimization process, the weight factor takes the value $w_{\max}$. The crossover and replacement occur as with a plain DE, see Subsection 2.1.

2.7 Self-adaptive Differential Evolution with Neighborhood Search

The Self-Adaptive Differential Evolution with Neighborhood Search (SaNSDE) [42] is a combination of two different algorithms: Differential Evolution with Neighborhood Search introduced in [41] and Self-Adaptive Differential Evolution (SADE) introduced in [30].

The SaNSDE modifies the DE in the following way. When the mutation is performed, the SaNSDE alternates, with a probabilistic scheme, two mutation strategies. The first strategy is the so called DE/rand/1 shown in eq. (1) while the second one is the DE/current to best/2 characterized by the formula:

$$x'_{off} = x_i + F\left(x_{p-best} - x_i\right) + F\left(x_r - x_s\right), \tag{10}$$

where x_{p_best} is the individual displaying best performance in the population, x_r and x_s are two individuals pseudo-randomly selected.

The probability of selecting the mutation strategy DE/rand/1 is initially 0.5 and subsequently updated (every 50 generations) by:

$$\rho = \frac{s_1(s_2 + f_2)}{s_2(s_1 + f_1) + s_1(s_2 + f_2)}, \tag{11}$$

where s_1, s_2, f_1, f_2 are respectively the number of successful and unsuccessful attempts at generating offspring by the two mutation strategy under investigation. More specifically, s_1 is the number of times the DE/rand/1 led to an offspring outperforming the corresponding parent, s_2 is the number of times the DE/current to best/2 led to an offspring outperforming the corresponding parent, f_1 is the number of unsuccessful attempts at generating an offspring by DE/rand/1 and f_2 is the number of unsuccessful attempts at generating an offspring by DE/current to best/2. The probability of selecting the mutation strategy DE/current to best/2 is given by $1 - \rho$.

A self adaptation of the scale factor F is also performed. For each individual a scale factor F_i is associated. Each scale factor is updated according to:

$$F_i = \begin{cases} N_i\left(0.5, 0.3\right) & \text{if} \quad rand < F_\rho \\ \delta_i & \text{otherwise} \end{cases} \tag{12}$$

where $N_i\left(0.5, 0.3\right)$ is a number sampled normal distribution with mean value of 0.5 and standard deviation equal to 0.3, *rand* is a pseudo-random number sampled from a uniform distribution, δ_i is a random sample from the Cauchy distribution with a scale parameter equal to 1, F_ρ is a probability constructed with the same logic as eq. (11) but related to the success of the scale factor.

In order to perform the crossover, the SaNSDE employs the weighted crossover rate self-adaptation. With each individual a crossover rate CR_i

is associated. Every five generations new CR_i values for each individual are generated for the new population by means of the following equation:

$$CR_i = N\left(CRm, 0.1\right). \tag{13}$$

The value CRm is initially set equal to 0.5 and then updated every 25 generations according to:

$$CRm = \sum_{k=1}^{|CRrec|} w_k CRrec\left(k\right) \tag{14}$$

where $CRrec$ is a vector containing the CR_i values which contributed to the generation of a successful offspring (individual which outperform its parent x_i). Each weight factor w_k is calculated as:

$$w_k = \frac{\Delta f_{rec}\left(k\right)}{\sum_{k=1}^{|\Delta f_{rec}|} \Delta f_{rec}\left(k\right)} \tag{15}$$

where $\Delta f_{rec}\left(k\right)$ is the enhancement in that fitness value corresponding to the application of $CRrec\left(k\right)$.

All remaining operations are performed as shown in Subsection 2.1.

3 Fitness Diversity Adaptation in Evolutionary Algorithms

As mentioned in [10] Exploration and Exploitation are two cornerstones in EAs and a proper balance of these properties seems to be the basic principle for algorithmic success. Unfortunately, every optimization problem has its own features, and thus, this balance must be designed taking into account the class of fitness landscapes which should be handled. In addition, as highlighted in [11], an EA is implicitly dynamic. Thus, in order to design a highly efficient algorithm, the explorative/exploitative pressure should vary over time and adapt to the demands of the optimization process while the search is performed.

A properly designed optimization algorithm should, in principle, be able to explore the decision space and detect the optimal basin of attraction, eventually converging to the global optimum. Unfortunately, the location of the global optimum and optimal basin of attraction is in general unknown a priori and it is impossible to even know during the optimization process whether one candidate solution has fallen within the optimal basin of attraction.

Nevertheless, the behavior of the algorithm (on a fitness landscape) can be observed and indirectly measured: if the population contains a high variety of genotypes (highly diverse), the algorithm explores the decision space and

hopefully detects new promising solutions; if the population contains a low variety of genotypes, the algorithm exploits a search direction and converges to a solution. The concept of diversity can then be used to monitor algorithmic behavior and prevent undesired stagnation and premature convergence situations. By means of a parameter adjustment, the algorithm can increase exploitation if the algorithm seems to be too explorative (checking a huge amount of solutions without improving upon the current best) or conversely increase exploration if the algorithm tends to lose the diversity and converge to a suboptimal solution (see for example [14], [11], and [9]). This logic has been implemented in various ways during the latest decades and has been widely used for local search coordination in Memetic Algorithms since their early definition in [20].

A not so trivial problem is how to efficiently measure the population diversity and how to make use of this information for actually obtaining an algorithm which adequately responds to variations. Measurement of the genotypical diversity presents the problem that a comparison among vectors of numbers is somehow required (e.g., distance between pairs of vectors) and a table expressing this comparison is generated; this operation can be computationally very expensive if the fitness is highly multi-variate.

An indirect way to measure the population diversity is through its fitness values. Measurement of the fitness diversity requires handling of only one vector of numbers which is composed of the fitness values of each individual of the population. On the other hand, the presence of plateaus and saddle points can give an inaccurate estimation of the distribution of points in the decision space. Fortunately, this limitation of the fitness diversity schemes is not very severe when this diversity is used for making an adaptive choice on the explorative/ exploitative countermeasures. If fitness diversity is high, the solutions are somehow spread out in the decision space, the algorithm is exploring new search directions and a higher exploitative pressure can help in choosing the most promising search direction; if the fitness diversity is low, either the algorithm is converging towards a solution or the entire population is contained in a plateau or a saddle; in both cases an increase in exploration is required in order to detect new promising solutions outside the current basin of attraction or plateau (or saddle).

The fitness diversity can thus be efficiently employed to measure the state of the algorithm and apply proper countermeasures to avoid stagnation and premature convergence, resulting in a high performance algorithm. In order to use information related to the fitness diversity, an index which estimates the diversity must be defined. In literature, several proposals have been made. Indicating with f_{best}, f_{avg} and f_{worst} respectively the best, average and worst fitness over the individuals of the population, in [4] and [23] the following index has been proposed:

$$\xi = \min \left\{ \left| \frac{f_{best} - f_{avg}}{f_{best}} \right|, 1 \right\} \tag{16}$$

In [22] and [24] the following parameter is given:

$$\psi = 1 - \left| \frac{f_{avg} - f_{best}}{f_{worst} - f_{best}} \right| \tag{17}$$

In [37] the fitness diversity has been measured by means of the following index:

$$\nu = \min \left\{ \frac{\sigma_f}{|f_{avg}|}, 1 \right\}, \tag{18}$$

where σ_f is the standard deviation over the fitness values of individuals of the population.

In [5] the following parameter is used:

$$\chi = \frac{|f_{best} - f_{avg}|}{max\,|f_{best} - f_{avg}|_k} \tag{19}$$

where f_{best} and f_{avg} are the fitness values of, respectively, the best and average individuals of the population. $max\,|f_{best} - f_{avg}|_k$ is the maximum difference observed (e.g., at the k^{th} generation), beginning from the start of the optimization process.

It must be noted that the four control parameters shown above can take values in the interval $[0, 1]$. These limit conditions 0 and 1 mean no fitness diversity and high fitness diversity respectively. Although all the above mentioned parameters measure fitness diversity of the population, the way this diversity is measured and scored differ much according to the index employed.

Although an in depth analysis of the structure of fitness diversity indexes is beyond the scope of this chapter, it is interesting to note that these parameters can be seen as an answer to four distinct questions related to the algorithms' state during the run. More specifically, ξ can be seen as the answer to the question "How close is the average fitness to the best one?"; ψ is the answer to the question "If we sort all fitness values over a line, which position is occupied by the average fitness?"; ν is the answer to the question "How sparse are the fitness values within the population?"; χ is the answer to the question "How much better is the best individual than the average fitness of the population with respect to the history of the optimization process?".

4 Fitness Diversity Self-adaptive Differential Evolution

This chapter aims to propose an efficient modification of DE which includes within its structure the fitness diversity logic in order to control explorative/exploitative pressure and thus improve upon the DE performance. The proposed algorithm, namely Fitness Diversity Self-Adaptive Differential Evolution (FDSADE) consists of the following steps.

4.1 Initial Sampling and Encoding of the Solutions

An initial sampling of S_{pop} individuals is executed pseudo-randomly with a uniform distribution function. As for the algorithm in [2], each individual x_i is composed of its genotype over the decision space D and its control parameters F_i and CR_i pseudo-randomly sampled between 0 and 1:

$$x_i = \langle x_{i,1}, x_{i,2}, ..., x_{i,j}, ...x_{i,n}, F_i, CR_i \rangle .$$

The fitness of each individual is calculated and the solutions are sorted according to their fitness values from best to worst.

4.2 Fitness Diversity Self-adaptation

At each generation the following fitness diversity index is calculated:

$$\phi = \frac{\sigma_f}{|f_{worst} - f_{best}|} \tag{20}$$

where σ_f is the standard deviation of fitness values over individuals of the populations, f_{worst} and f_{best} are the worst and best fitness values, respectively, of the population individuals.

Analogous to the other fitness diversity indexes listed in section 3, ϕ varies between 0 and 1. When the fitness diversity is high, $\phi \approx 1$; on the contrary when the fitness diversity is low, $\phi \approx 0$. The index ϕ can be seen as a combination of ν in formula (18) and ψ in formula (17) because it represents the distribution of fitness values over individuals of the population with respect to its range of variability. In other words, ϕ is the answer to the question "How sparse are the fitness values with respect to the range of fitness variability at the current generation?". Employment of the standard deviation in the numerator in formula (20) is due to the fact that a DE framework tends to generate an individual with performance significantly above the average (see [37] and [5]) and efficiently continues optimization for several generations. In this sense, an estimation of the fitness diversity of a DE population by means of the difference between best and average fitness values can return a misleading result and each value must be taken into account. Regarding the denominator in formula (20), a normalization to the range of variability of the current population makes the index co-domain invariant (unlike ν in formula (18)) and its estimation is not affected, for example by adding an offset to the fitness function. Thus, the index ϕ can be successfully employed, within a DE framework, on problems of various kinds.

The control parameter F_i is then updated according to the scheme:

$$F_i = \begin{cases} F_l + F_u rand_1, \text{ if } rand_2 < K\,(1 - \phi) \\ F_i, \qquad\qquad \text{otherwise} \end{cases} \tag{21}$$

where $rand_1$ and $rand_2$ are pseudo-random numbers generated by means of the uniform distribution, F_l and F_u are the same constant values shown in formula (4) for the SACPDE.

Analogously, the control parameter CR_i is updated according to the scheme:

$$CR_i = \begin{cases} rand_3, \text{ if } rand_4 < K\,(1-\phi) \\ CR_i, \qquad\qquad \text{otherwise} \end{cases} \tag{22}$$

where $rand_3$ and $rand_4$ are pseudo-random numbers generated by means of the uniform distribution. For both equations (21) and (22), the constant value K represents the maximum update probability of the parameters.

In other words, the proposed algorithm hybridizes the self-adaptive scheme proposed in [2] with the fitness diversity logic in order to obtain a high performance DE algorithm. The main idea behind the proposed self-adaptation is that in high diversity conditions ($\phi \approx 1$), the solutions x_i should tend to keep the control parameters F_i and CR_i unvaried and thus exploit the current potential search. On the contrary, when the diversity condition is low ($\phi \approx 0$), the algorithm should try to better explore the decision space by frequently changing intensity of the mutation move F_i and updating the recombination rate CR_i. The proposed logic is thus similar to the fitness diversity based activation of a local search in a MA (see e.g., [22]): if the algorithms need to exploit the genotype no changes to the evolutionary framework are necessary, if the algorithm has poor diversity a change in the exploratory logic and exploration of new search perspectives are recommended (see [15]).

4.3 Recombination Operators

Recombination operations, i.e., mutation and crossover, occur in the FD-SADE, similar to operations performed in the standard DE. For each solution x_i, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population and a provisional offspring x'_{off} is generated by mutation:

$$x'_{off} = x_t + F_i(x_r - x_s) \tag{23}$$

where F_i is the scale factor corresponding to solution x_i. Each gene of the new individual x'_{off} is then switched with the corresponding gene of x_i with the corresponding uniform probability CR_i and the final offspring x_{off} is generated:

$$x_{off,j} = \begin{cases} x_{i,j} & \text{if} \quad rand\,(0,1) < CR_i \\ x'_{off,j} & otherwise \end{cases} \tag{24}$$

4.4 Adaptive Population Size and Selection

When S_{pop} offspring individuals are generated, the offspring x_{off} is evaluated and, according to standard DE logic, replaces x_i if and only if $f(x_{off}) < f(x_i)$; otherwise no replacement occurs.

generate S_{pop} individuals of the initial population with related parameters pseudo-randomly;
while budget condition
  for $i = 1 : S_{pop}$
    compute $f(x_i)$;
  end-for
  sort the population;
  compute $\phi = \frac{\sigma_f}{|f_{worst} - f_{best}|}$;
  for $i = 1 : S_{pop}$
    F_i update
    generate $rand_1$ and $rand_2$;
    $F_i = \begin{cases} F_l + F_u rand_1, \text{ if } rand_2 < K(1-\phi) \\ F_i, \qquad\qquad \text{otherwise} \end{cases}$;
    mutation
    select three individuals x_r, x_s, and x_t;
    compute $x'_{off} = x_t + F_i(x_r - x_s)$;
    CR_i update
    generate $rand_3$ and $rand_4$;
    $CR_i = \begin{cases} rand_3, \text{ if } rand_4 < K(1-\phi) \\ CR_i, \qquad \text{otherwise} \end{cases}$
    crossover
    $x_{off} = x'_{off}$;
    for $j = 1 : n$
      generate $rand(0,1)$;
      if $rand(0,1) < CR_i$
        $x_{off,j} = x_{i,j}$;
      end-if
    end-for
    selection
    if $f(x_{off}) < f(x_i)$
      $x_i = x_{off}$;
    end-if
  end-for
  population size adjustment
  compute $\phi = \frac{\sigma_f}{|f_{worst} - f_{best}|}$;
  $S_{pop}^{old} = S_{pop}$;
  compute $S_{pop} = S_{pop}^{f} + S_{pop}^{v}(1-\phi)$;
  if $S_{pop} < S_{pop}^{old}$
    select the S_{pop} individuals with the best performance;
  else duplicate in the population $S_{pop} - S_{pop}^{old}$
      solutions with the best performance;
  end-if;
end-while

Fig. 5 FDSADE pseudocode

The parameter ϕ is then calculated as shown in formula (20) and population size is adjusted for the subsequent generation according to the equation:

$$S_{pop} = S^f_{pop} + S^v_{pop}\,(1 - \phi) \tag{25}$$

where S^f_{pop} stands for fixed population size and is the minimum value of the population size, S^v_{pop} stands for variable population size and is the maximum value of the variable contribution of the population size. The new value of S_{pop} is then employed for the subsequent generation: if population size is reduced, only those S_{pop} individuals having the best performance are considered for the next steps; if the population is expanded, the best solutions are duplicated in order to have a population composed of S_{pop} individuals.

The meaning of this variable population size is that in high diversity conditions the algorithm should shrink the population and exploit the available genotypes, in low diversity conditions the algorithm should enlarge the population and through the recombination mechanism focus on detecting new promising search directions. In this way the algorithm should be able to adapt to necessities of the fitness landscape and dynamically balance the explorative/exploitative necessities. Similar approaches have been proposed in the literature, e.g., [4], [23], [21] and [24].

For sake of clarity the pseudo-code illustrating the working principles of the FDSADE is given in Fig. 5.

5 Numerical Results

The FDSADE has been tested on a set of various test problems and compared with a plain DE [33], SACPDE [2], DEahcSPX [26], OBDE [31], the DEGL [8], and SaNSDE [42].

- The DE has been run with $F = 0.7$ and $CR = 0.7$ in accordance to the suggestions given in [44].
- The SACPDE has been run, with reference to the formulas (4) and (5), with $F_l = 0.1$, $F_u = 0.9$, $\tau_1 = 0.1$, and $\tau_2 = 0.1$ as shown in [2].
- The DEahcSPX has been run with $F = 0.7$ and $CR = 0.7$ (in accordance with [44]) and, with reference to Fig. 3, $n_p = 10$.
- The DEGL has been run, with reference to formulas (6), (7), (8), and (9), with $k = 5$, $\alpha = \beta = 0.8$, $w_{\min} = 0.4$, $w_{\max} = 0.8$, and $CR = 0.7$
- The SaNSDE has been run as explained in Subsection 2.7
- The FDSADE has been run with reference to formulas (21) and (22), $F_l = 0.1$, $F_u = 0.9$ as suggested in [2] and $K = 0.3$.

Regarding the population size, the FDSADE has been run with $S^f_{pop} = 10$, $S^v_{pop} = 40$ (see formula (25)) while all the other algorithms have been run with $S_{pop} = 30$. Each algorithm has been run for 30 independent runs, 50000 fitness evaluations each run.

The choice of $K = 0.3$ has been empirically performed after having carried out an analysis of the algorithmic behavior with dependance on K. The main result of our analysis is that the FDSADE has a high algorithmic performance for chosen K values in the interval $[0.1, 0.7]$ and performance of the FDSADE is not very sensitive to the chosen K value within this interval. Nevertheless, it has been observed that the best performance is obtained in correspondence to $K = 0.3$. Fig. 6 shows an example of the FDSADE average performance (over 30 runs) with dependence on various values of K. Fig. 6 refers to the Michalewicz function in 30 dimensions.

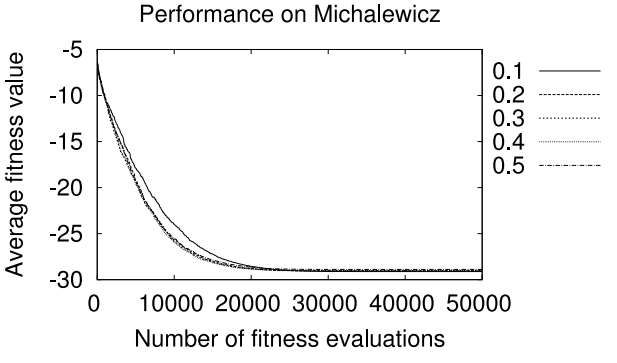


Fig. 6 Algorithmic performance in dependance on K

The test problems under investigation are listed in Table 1. It should be remarked that some rotated problems have been added to our benchmark set. The rotated problems are obtained by means of multiplication of the vector of variables to a randomly generated orthogonal rotation matrix. The test problem indicated with "Tirronen" is generated by means of a novel test function proposed in this chapter. This test function has been included in order to obtain a highly multi-modal landscape which contains a global minimum in an asymmetrical position and a pattern that changes towards the minimum (unlike Rastrigin or Ackley, whose pattern is orthogonal to the axes throughout the entire landscape). Fig. 7 shows the Tirronen function in two dimensions.

Table 2 shows the average final results (after 50000 fitness evaluations) and the related standard deviation, calculated over the 30 available runs. The best results are highlighted in bold face.

It can be noted that only for the Easom and Camelback functions do all algorithms converge to the same value. For the other eighteen test problems, one of the algorithms outperforms all others in terms of final value. Table 2 shows that the DEGL seems to be a very competitive algorithm since it produced the best performance in ten cases; the proposed FDSADE reached the best final value in seven cases out of twenty under analysis, the DE

Table 1 Test Problems

Test Problem	n	**Function**	**Decision Space**
Ackley	30	$-20 + e + \exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $- \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)x_i\right)$	$[-1,1]^n$
Alpine	30	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1x_i \rvert$	$[-10,10]^n$
Camelback	2	$4x_1^2 - 2.1x_1^2 + \frac{x_1^6}{3} + x_1x_2 - 4x_2^2 + 4x_2^4$	
DeJong	30	$\lvert\lvert x \rvert\rvert^2$	$[-5.12,5.12]^n$
DropWave	30	$-\frac{1+\cos\left(12\sqrt{\lvert\lvert x \rvert\rvert^2}\right)}{\frac{1}{2}\lvert\lvert x \rvert\rvert^2+2}$	$[-5.12,5.12]^n$
Easom	2	$\cos x_1 \cos x_2 \exp\left(-(x_1-\pi)^2 - (x_2-\pi)^2\right)$	$[-100,100]^n$
Griewangk	30	$\frac{\lvert\lvert x \rvert\rvert^2}{4000} - \prod_{i=0}^{n}\cos\frac{x_i}{\sqrt{i}} + 1$	$[-600,600]^n$
Michalewicz	30	$-\sum_{i=1}^{n}\sin x_i \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)$	$[0,\pi]^n$
Pathological	30	$\sum_{i=1}^{n-1}\left(0.5 + \frac{\sin^2\left(\sqrt{100x_i^2+x_{i+1}^2}-0.5\right)}{1+0.001*\left(x_i^2-2x_ix_{i+1}+x_{i+1}^2\right)^2}\right)$	$[-100,100]^n$
Rosenbrock	30	$\sum_{i=0}^{n-1}\left(\left(x_{n+1}-x_i^2\right)^2 + (1-x)^2\right)$	$[-2.048,2.048]^n$
Rastrigin	30	$10n + \sum_{i=0}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12,5.12]^n$
Schwefel	30	$\sum_{i=1}^{n} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500,500]^n$
Sum of powers	30	$\sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$	$[-1,1]^n$
Tirronen	30	$3\exp\left(-\frac{\lvert\lvert x \rvert\rvert^2}{10n}\right) - 10\exp\left(-8\lvert\lvert x \rvert\rvert^2\right)$ $+\frac{2.5}{n}\sum_{i=1}^{n}\cos\left(5(x_i+(1+i \mod 2))cos(\lvert\lvert x \rvert\rvert)\right)$	$[-10,5]^n$
Whitley	30	$\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\frac{y_{i,j}^2}{4000} - \cos\left(y_{i,j}\right) + 1\right)$, where $y_{i,j} = \left(100(x_j-x_i)^2 + (1-x_i)^2\right)^2$	$[-100,100]^n$
Zakharov	30	$\lvert\lvert x \rvert\rvert^2 + \left(\sum_{i=1}^{n}\frac{ix_1}{2}\right)^2 + \left(\sum_{i=1}^{n}\frac{ix_1}{2}x_i\right)^4$	$[-5,10]^n$

reached the best final value in five cases, the SaNSDE and the SACPDE reached the best final values for four test problems and the DEahcSPX only with the Easom and Camelback functions. In addition, it can be seen that when the FDSADE does not reach the best value, it will in any case often

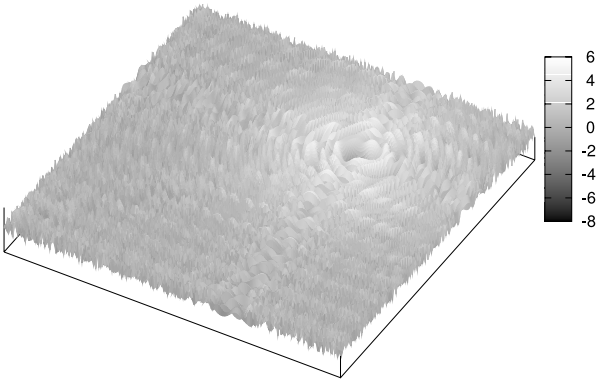


Fig. 7 Tirronen function

return a solution with a high performance (see e.g., the values in De Jong or Rotated Rastrigin). Finally, it must be highlighted that the FDSADE never obtained the worst results over the twenty test problems under analysis and in most cases offers a competitive performance in terms of final solution.

In order to prove statistical significance of the results, the Student's t-test has been applied according to the description given in [25] for a confidence level of 0.95. The final values obtained by the FDSADE have been compared to the final value returned by each algorithm used as a benchmark. Table 3 shows results of the test. Indicated with "+" is the case when the FDSADE statistically outperforms, for the corresponding test problem, the algorithm mentioned in the column; indicated with "=" is the case when pairwise comparison leads to success of the t-test i.e., the two algorithms have the same performance; indicated with "-" is the case when the FDSADE is outperformed.

Table 2 and Table 3 show that DEGL performs the best, in terms of final values, among all the algorithms considered here. However, it must be noticed from Table 3 that the FDSADE is outperformed in only eighteen cases out of the hundred-twenty pairwise comparisons considered, the FDSADE has the same algorithmic performance as the other algorithms in twenty-six cases and outperforms the other algorithms in fifty-six cases. It can thus be concluded that the FDSADE is outperformed in only 15% of the cases and outperforms the other algorithms in 56% of the cases. Therefore, the FDSADE, in a statistical sense, offers good performance in detecting a final solution with high quality and is competitive with modern DE based algorithms. Considering that the experimental setup is composed of a very diverse set of test problems (e.g., some functions are uni-modal, others are highly multi-modal), the FDSADE seems to have a high performance with various kinds of problems. Finally, attention must be paid in observing the comparison between FDSADE and SACPDE since the FDSADE has the same self-adaptive structure of the SACPDE and the same update logic of the control parameters.

Table 2 Final Values, Average ± Standard Deviation

Test Prob.	DE	SACPDE	DEahcSPX	DEGL	SaNSDE	FDSADE
Ackley	6.17e-10±1.82e-10	3.16e-07±1.70e-06	9.42e-02±6.24e-02	**7.19e-15±1.07e-15**	6.21e-02±2.32e-01	1.75e-12±7.22e-12
Alpine	3.35e-02±1.87e-02	3.40e-05±7.30e-05	1.28e+00±1.05e+00	**1.12e-10±5.92e-10**	5.32e-03±2.87e-02	1.24e-04±4.88e-04
Camelback	**-1.03e+00±6.66e-16**	**-1.03e+00±6.66e-16**	-1.03e+00±5.73e-16	**-1.03e+00±6.66e-16**	**-1.03e+00±6.66e-16**	**-1.03e+00±6.66e-16**
De Jong	1.59e-57±1.94e-57	**4.25e-86±1.57e-85**	2.24e-02±4.73e-02	9.66e-50±1.34e-49	9.92e-19±5.16e-18	2.26e-66±2.92e-66
Dropwave	-5.98e-01±4.72e-02	-2.50e-01±6.70e-02	-1.82e-01±4.54e-02	**-7.91e-01±2.70e-02**	-1.73e-01±6.66e-02	-2.60e-01±6.90e-02
Easom	**-1.00e+00±0.00e+00**	**-1.00e+00±0.00e+00**	-1.00e+00±7.21e-24	**-1.00e+00±0.00e+00**	**-1.00e+00±0.00e+00**	**-1.00e+00±0.00e+00**
Griewangk	**2.27e-13±5.36e-13**	3.90e-02±6.13e-02	3.70e+00±3.30e+00	5.75e-04±2.18e-03	4.63e-01±9.26e-01	9.00e-03±1.57e-02
Rot. Griew.	**1.86e-03±4.14e-03**	6.25e-03±8.48e-03	3.71e+00±2.67e+00	2.79e-03±5.27e-03	9.51e-03±1.19e-02	2.71e-03±4.73e-03
Michalew.	-2.89e+01±1.97e-01	-2.89e+01±2.76e-01	-1.48e+01±1.21e+00	-2.10e+01±5.96e-01	-2.45e+01±1.27e+00	**-2.90e+01±2.40e-01**
Rot. Mich.	-1.15e+01±5.85e-01	-1.91e+01±2.16e+00	-1.07e+01±7.98e-01	-9.46e+00±6.47e-01	-1.37e+01±1.50e+00	**-2.14e+01±2.26e+00**
Patholog.	2.45e+01±2.46e-05	2.45e+01±3.20e-06	2.45e+01±4.59e-04	**1.45e+01±5.79e-07**	1.45e+01±3.36e-06	2.45e+01±1.66e-06
Rastrigin	**8.87e+00±3.19e+00**	1.23e+01±3.55e+00	1.25e+02±1.72e+01	5.68e+01±8.56e+00	6.88e+01±1.87e+01	1.13e+01±3.73e+00
Rot. Rast.	4.36e+02±1.91e+01	8.44e+01±1.81e+01	4.00e+02±1.54e+01	1.67e+02±1.56e+01	**7.28e+01±2.37e+01**	9.13e+01±1.90e+01
Rosenbrock	7.01e-10±5.34e-10	4.21e-04±1.32e-03	3.42e+00±1.06e+00	**3.69e-20±1.15e-19**	1.34e-07±2.24e-07	1.16e-05±1.77e-05
Schwefel	-1.19e+04±2.75e+02	-1.14e+04±3.18e+02	-5.62e+03±5.16e+02	-1.04e+04±1.05e+03	-9.39e+03±4.73e+02	**-1.19e+04±2.85e+02**
Rot. Schw.	-7.36e+03±3.61e+02	-1.11e+04±8.25e+02	-5.35e+03±4.29e+02	-6.56e+03±5.30e+02	-8.56e+03±6.63e+02	**-1.41e+04±8.51e+02**
Sum pow.	9.21e-36±1.67e-35	2.52e-07±7.68e-07	1.04e-04±3.97e-04	**3.98e-66±1.03e-65**	6.24e-31±2.29e-30	3.25e-08±1.33e-07
Tirronen	-1.86e+00±5.92e-02	-2.06e+00±1.47e-01	-1.07e+00±9.34e-02	-1.98e+00±6.16e-02	-2.03e+00±1.92e-01	**-2.15e+00±7.88e-02**
Whitley	1.60e+03±1.20e+02	1.66e+12±5.59e+12	1.00e+13±6.93e+14	**3.23e+02±1.43e+02**	3.06e+03±3.27e+03	1.85e+12±6.57e+12
Zakharov	5.90e+02±7.64e+01	2.07e+01±1.30e+01	1.77e+02±5.54e+01	**2.20e+00±1.10e+00**	4.07e+01±5.05e+01	2.20e+01±1.05e+01

Table 3 Results of the Student's t-test

Test Problem	DE	SACPDE	DEahcSPX	DEGL	SaNSDE
Ackley	+	=	+	=	+
Alpine	+	=	+	−	=
Camelback	=	=	=	=	=
De Jong	=	=	+	=	=
Dropwave	−	=	+	−	+
Easom	=	=	=	=	=
Griewangk	−	+	+	−	+
Rot. Griewangk	=	+	+	=	+
Michalewicz	+	+	+	+	+
Rot. Michalewicz	+	+	+	+	+
Pathological	+	+	+	−	−
Rastrigin	−	=	+	+	+
Rot. Rastrigin	+	−	+	+	−
Rosenbrock	−	+	+	−	−
Schwefel	=	+	+	+	+
Rot. Schwefel	+	+	+	+	+
Sum of powers	−	+	+	−	−
Tirronen	+	+	+	+	+
Whitely	−	=	=	−	−
Zakharov	+	=	+	−	+

As shown in Section 4, the FDSADE integrates, in addition to the SACPDE, the fitness diversity philosophy and fitness diversity based variable population size. Integration of the fitness diversity seems to be very promising because, as shown in Table 3, it leads to an improvement of performance in ten cases, the same performance in nine cases and a worse performance in only one case. This analysis confirms that the fitness diversity adaptation can be an efficient instrument in enhancing the effectiveness of an algorithm.

In order to carry out a numerical comparison of the convergence speed performance, for each test problem the average final fitness value returned by the best performing algorithm G has been considered. Subsequently, the average fitness value at the beginning of the optimization process J has also been computed. The threshold value $THR = J - 0.95(G - J)$ has then been calculated. The value THR represents 95% of the decay in the fitness value in the best performing algorithms fitness value. If an algorithm succeeds during a certain run to reach the value THR, the run is said to be successful. For each test problem the average amount of fitness evaluations $\bar{ne}$ required for each algorithm to reach THR has been computed. Subsequently, the Q-test (Q stands for Quality) described in [12] has been applied. For each test problem and each algorithm, the Q measure is computed as $Q = \frac{\bar{ne}}{R}$ where the robustness R is the percentage of successful runs. It is clear that for each test problem the smallest value means the best performance in terms of convergence speed. The value inf means that $R = 0$, i.e. the algorithm never

reached the THR. Table 4 shows the Q values in 30 dimensions. The best results are highlighted in bold face.

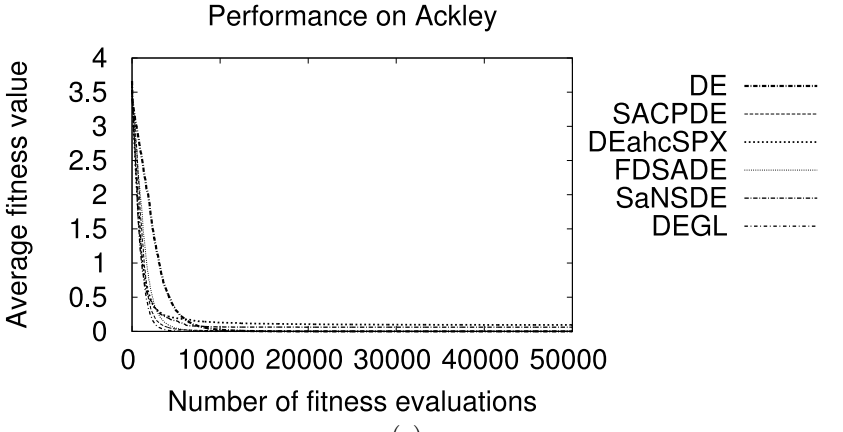
Figures 8 show the average performance trend (over 30 independent runs) of the six algorithms under analysis over some of the test problems listed in Table 1.

It can be observed that the FDSADE fails in detecting the optimum, with respect to the others, only in the case of the Dropwave and Whitley functions. In all other cases the FDSADE displays a high performance; the proposed algorithm is either competitive with another algorithm of the benchmark (see e.g., Rotated Griewangk, Michalewicz, Pathological, and Rotated Rastrigin) or has extraordinarily high performance as shown with Rotated Michalewicz, Schwefel, and Rotated Schwefel. Performance in terms of convergence speed is also competitive with the other algorithms in most of the cases analyzed.

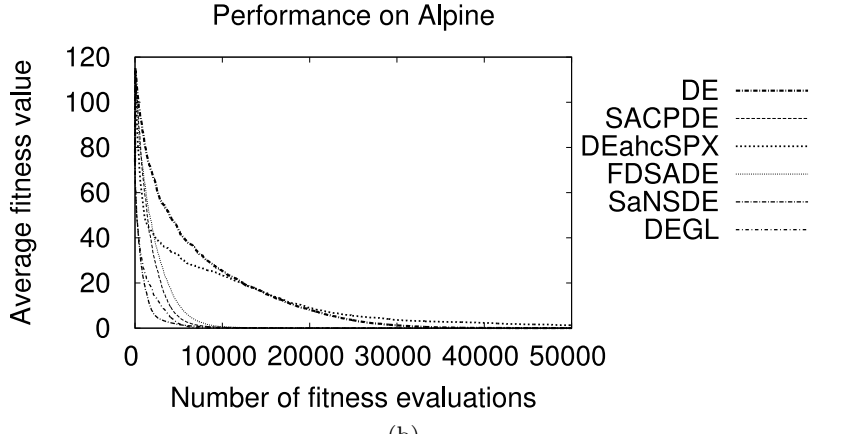
Numerical results in Table 4 show that although the FDSADE does not always have the best performance in terms of convergence velocity, it is in most cases very competitive with the algorithm that has the best performance. In other words, the other algorithms seem to have a high convergence velocity performance with some test problems and a poor performance with others. On the contrary, the FDSADE demonstrates a performance close to the best in almost all test problems. In addition, it must be remarked that the FDSADE does not reach THR in only two cases (Dropwave and Pathological

Table 4 Results of the Q-test

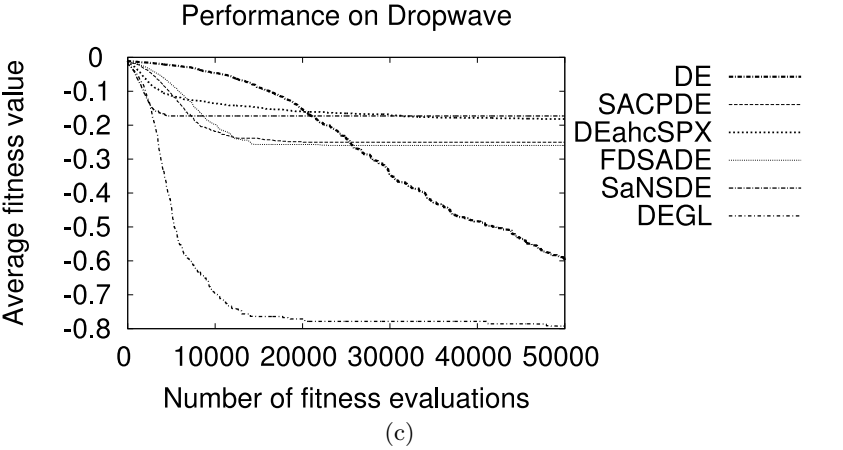
Test Problem	DE	SACPDE	DEahcSPX	DEGL	SaNSDE	FDSADE
Ackley	6.0e+01	2.8e+01	6.4e+01	**2.3e+01**	3.4e+01	3.4e+01
Alpine	2.3e+02	5.0e+01	2.4e+02	4.1e+01	**2.7e+01**	6.4e+01
Camelback	2.5e+00	2.5e+00	1.5e+01	2.4e+00	**1.6e+00**	2.6e+00
De Jong	8.0e+00	**5.3e+00**	7.0e+00	1.6e+01	2.6e+01	6.7e+00
Dropwave	inf	inf	inf	**1.3e+02**	inf	inf
Easom	1.7e+01	9.8e+00	8.6e+01	1.5e+01	**5.9e+00**	1.4e+01
Griewangk	3.5e+01	1.6e+01	1.4e+01	**1.3e+01**	1.7e+01	2.0e+01
Rotated Griewangk	3.4e+01	1.6e+01	1.4e+01	**1.3e+01**	1.5e+01	2.0e+01
Michalewicz	3.1e+02	**1.1e+02**	inf	inf	inf	1.4e+02
Rotated Michalewicz	inf	2.1e+03	inf	inf	inf	**5.2e+02**
Pathological	inf	inf	inf	**4.0e-01**	**4.0e-01**	inf
Rastrigin	2.8e+02	**5.7e+01**	inf	inf	inf	6.5e+01
Rotated Rastrigin	inf	2.5e+02	inf	inf	5.1e+01	**2.3e+02**
Tirronen	inf	7.4e+02	inf	6.4e+03	**1.7e+02**	4.9e+02
Rosenbrock	5.3e+01	**2.6e+01**	9.6e+01	1.9e+01	2.8e+01	3.1e+01
Schwefel	2.2e+02	1.3e+02	inf	1.8e+03	inf	**8.2e+01**
Rotated Schwefel	inf	inf	inf	inf	inf	**2.5e+02**
Sum of powers	2.1e+01	7.0e+00	5.2e+00	4.4e+00	**4.2e+00**	7.3e+00
Whitely	2.7e+01	8.1e+00	5.7e+00	3.6e+00	**3.2e+00**	9.7e+00
Zakharov	1.2e+00	5.7e+00	1.5e+01	**4.0e-01**	**4.0e-01**	1.5e+00



(a)

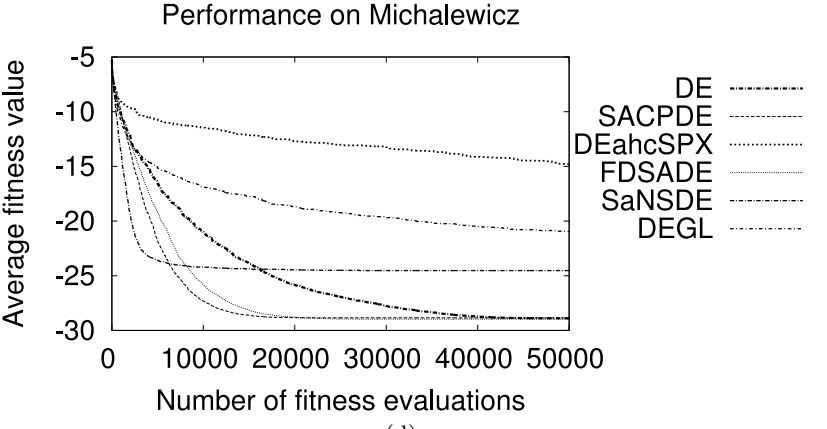


(b)

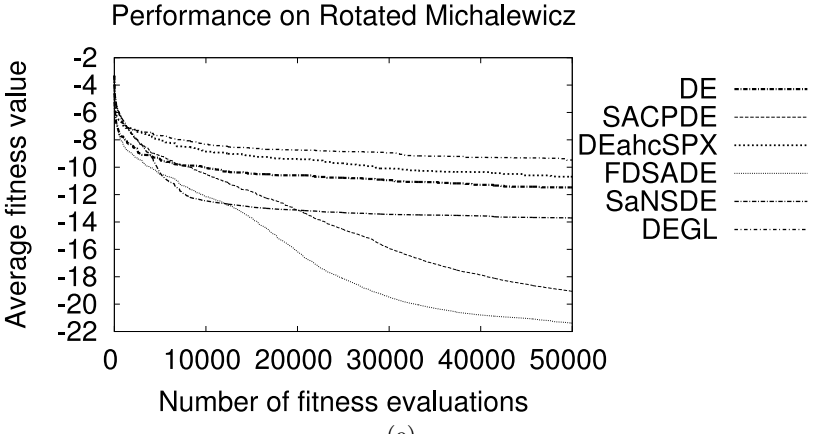


(c)

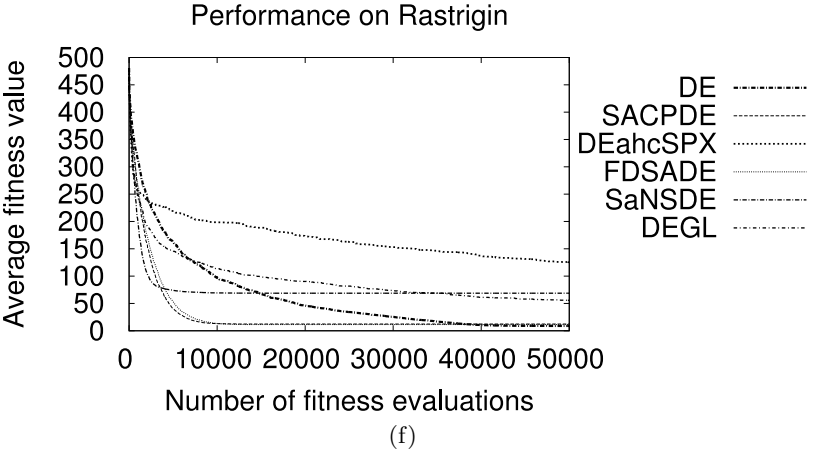
Fig. 8 Algorithmic performance over selected test problems



(d)

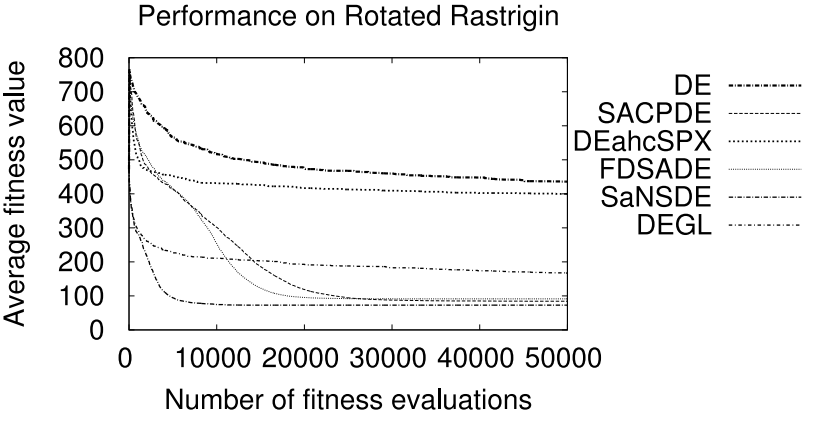


(e)

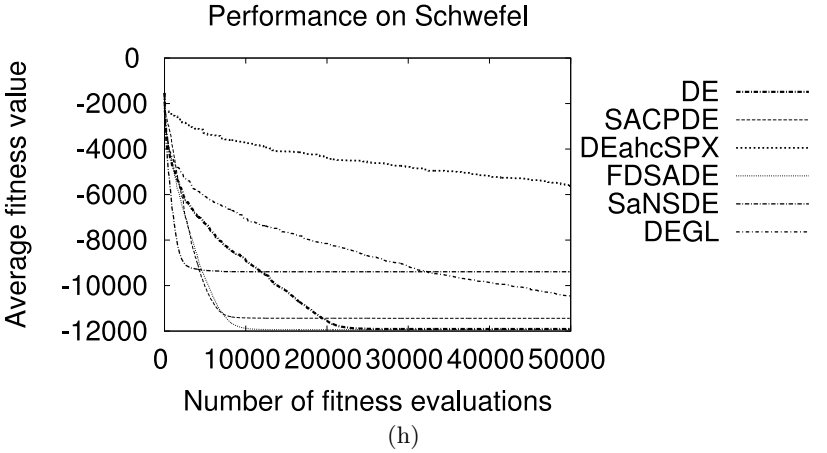


(f)

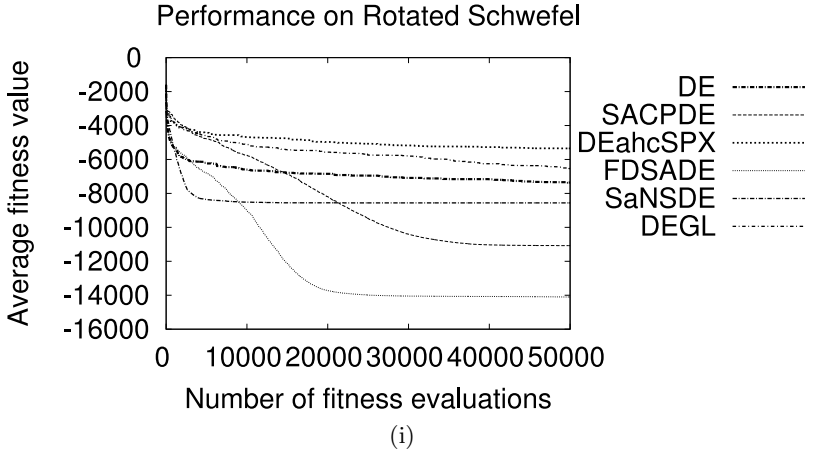
Fig. 8 (*continued*)



(g)



(h)



(i)

Fig. 8 (*continued*)

functions); in all other cases it reaches the threshold in at least one case. This result confirms that the FDSADE has a robust behavior over a set of various test problems. It can be seen that, in this sense, the FDSADE is the best algorithm (it is the algorithm with the fewest number of inf values) among the six under examination.

5.1 Numerical Results for (Relatively) Large Scale Problems

The previous six algorithms have also been run in order to minimize ten of the functions in Table 1 for $n = 100$. The algorithms have been run with the same parameter setting mentioned above except population size. The FDSADE has been run with $S^f_{pop} = 30$, $S^v_{pop} = 120$ while all the other algorithms with $S_{pop} = 100$. For each algorithm, 30 independent runs have been performed for 50000 fitness evaluations. Table 5 shows the average final results (after 50000 fitness evaluation) and the related standard deviation, calculated over the 30 available runs.

It must be observed that the algorithmic performance is dramatically different with respect to the numerical results in 30 dimensions. An important fact is that, in accordance with the No Free Lunch Theorem [40], the algorithmic performance in high dimensions is very problem-dependant, i.e., none of the algorithms examined seems to be, in general, clearly superior to the others. The performance comparison reported here is valid only for the parameter settings used in this chapter. The effect of variation of parameters has not been investigated.

However, it can be noticed that the DE and SACPDE never have the best performance in terms of quality of the final solution. Performance of the DEGL is not so promising as the one displayed in the low dimensional case. On the contrary, the SaNSDE seems to be rather promising for large scale optimization problems. The FDSADE is competitive with the SaNSDE and reaches quite good results in all the test problems considered.

Table 6 shows results of the Student's t-test in 100 dimensions.

The t-test in Table 6 shows that the FDSADE is outperformed nine times and has the same performance in only two cases out of the fifty pairwise comparisons considered. Thus, the FDSADE is outperformed in only 18% of the cases and outperforms the other algorithms in 78% of the cases. Finally, it must be observed that in 100 dimensions, the FDSADE either obtained results similar to the SACPDE or succeeded in improving upon the SACPDE performance, thus confirming the efficiency of the approach proposed.

Table 7 shows the Q-test results for the 100 dimension case.

Regarding convergence speed and robustness of the algorithms, since in the high dimensional space the various algorithms tend to reach very different values, for each test problem that algorithm which reaches final values with

Table 5 Final Values, Average ± Standard Deviation in 100 dimensions

Test Problem	DE	SACPDE	DEahcSPX	DEGL	SaNSDE	FDSADE
Rot. alpine	2.00e+02±9.59e+00	1.29e+02±1.38e+01	**6.83e-01±4.98e-01**	6.11e+00±7.93e+00	4.04e+01±9.14e+00	1.16e+01±6.78e+00
Dropwave	-9.71e-03±2.14e-04	-1.20e-02±0.00e+00	**-4.78e-01±0.00e+00**	-3.67e-01±0.00e+00	-4.91e-02±1.02e-02	-1.82e-01±6.72e-03
Michalewicz	-3.62e+01±0.00e+00	-2.82e+01±1.88e+00	-2.84e+01±4.95e-01	-3.65e+01±1.04e+00	**-7.24e+01±2.22e+00**	-5.66e+01±7.54e-01
Rot. Michalewicz	-1.51e+01±7.42e-01	-1.46e+01±7.88e-01	-1.63e+01±5.06e-01	-1.69e+01±1.16e+00	**-2.31e+01±5.79e+00**	-1.84e+01±1.46e+00
Rastrigin	9.44e+02±3.07e+01	5.34e+01±5.44e+01	8.07e+02±6.23e+01	6.57e+02±1.89e+01	2.80e+02±4.28e+01	**5.08e+01±8.82e+00**
Rot. Rastrigin	1.26e+03±3.29e+01	5.08e+02±4.80e+01	8.72e+02±3.81e+00	8.95e+02±2.65e+01	**2.87e+02±4.58e+01**	4.12e+02±7.62e+01
Rosenbrock	8.25e+01±7.88e+00	8.98e+01±1.33e+01	3.05e-01±6.24e-02	**1.41e-01±3.20e-02**	3.74e+00±1.78e+00	1.33e+00±1.87e-01
Schwefel	-1.72e+04±6.80e+02	-3.69e+03±1.31e+03	-9.18e+03±5.14e+02	-1.32e+04±5.42e+02	-2.91e+04±1.18e+03	**-3.99e+04±5.86e+02**
Rot. Schwefel	-1.03e+04±7.34e+02	-9.44e+03±6.07e+02	-8.56e+03±6.19e+02	-1.00e+04±6.80e+02	**-2.94e+04±1.54e+03**	-1.85e+04±3.74e+03
Zakharov	2.09e+03±1.20e+02	6.98e+10±3.56e+11	1.29e+03±1.92e+02	1.12e+03±9.45e+01	1.66e+03±2.39e+02	**5.53e+02±2.51e+02**

Table 6 Results of the Student's t-test in 100 dimensions

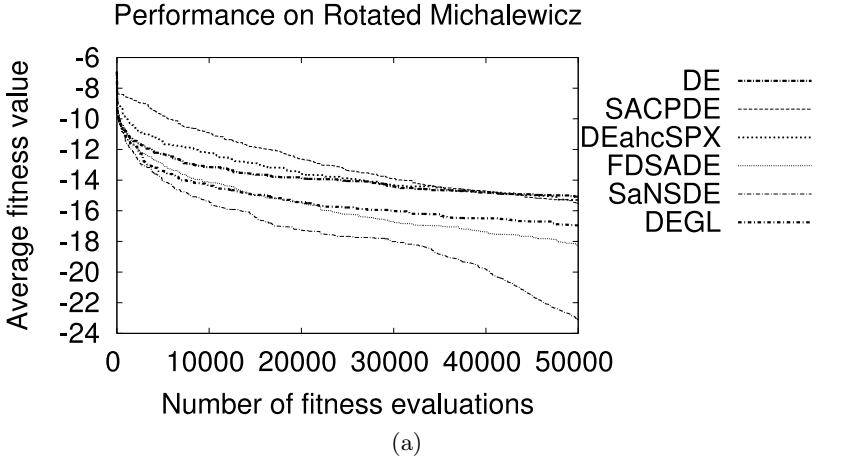
TestProblem	DE	SACPDE	DEahcSPX	DEGL	SaNSDE
Rotated alpine	+	+	−	−	+
Dropwave	+	+	−	−	+
Michalewicz	+	+	+	+	−
Rotated Michalewicz	+	+	+	+	−
Rastrigin	+	=	+	+	+
Rotated Rastrigin	+	+	+	+	−
Rosenbrock	+	+	−	−	+
Schwefel	+	+	+	+	+
Rotated Schwefel	+	+	+	+	−
Zakharov	+	=	+	+	+

Table 7 Results of the Q-test in 100 dimensions

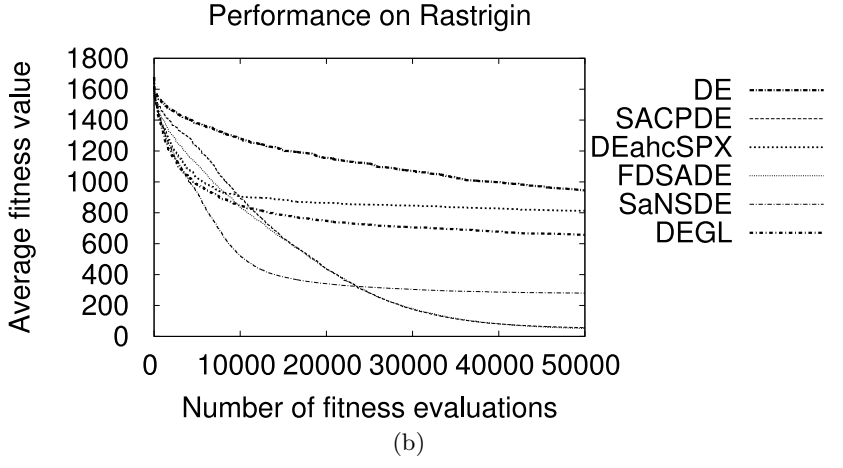
Test Problem	DE	SACPDE	DEahcSPX	DEGL	SaNSDE	FDSADE
Rotated alpine	inf	4.4e+02	**3.1e+02**	5.1e+02	inf	7.1e+02
Dropwave	inf	inf	**4.6e+03**	inf	inf	inf
Michalewicz	inf	inf	inf	inf	**4.2e+02**	inf
Rotated Michalewicz	inf	inf	inf	inf	**1.0e+03**	inf
Rastrigin	inf	**3.4e+02**	inf	inf	inf	**3.4e+02**
Rotated Rastrigin	inf	2.4e+03	inf	inf	**3.3e+02**	1.4e+03
Rosenbrock	inf	1.9e+02	1.4e+02	**1.2e+02**	1.7e+02	1.9e+02
Schwefel	inf	1.4e+04	inf	inf	inf	**3.9e+02**
Rotated Schwefel	inf	inf	inf	inf	**3.3e+02**	inf
Zakharov	5.1e+00	5.5e+01	2.9e+01	3.6e+00	**2.0e+00**	5.2e+00

a best performance is usually the one which has the best performance in terms of Q measures. In this sense, numerical results in Table 7 confirm the findings in Table 5 and Table 6, i.e., the SaNSDE has very good performance in highly dimensional problems and the FDSADE is, in any case, quite competitive. On the other hand, behavior of the algorithms in terms of robustness is worthwhile commenting on. As shown in Table 7, the DE succeeds in reaching a competitive value (reaches the threshold value) for only one test problem, the DEGL in three cases, the DEahcSPX in four cases, SACPDE, SaNSDE and FDSADE in six cases out of ten test problems considered. This fact means there is not an algorithm which has an extraordinarily high performance in terms of robustness among the ones considered. However the FDSADE maintains the good robustness performance of the SACPDE and tends to improve upon this in late stages of the optimization process.

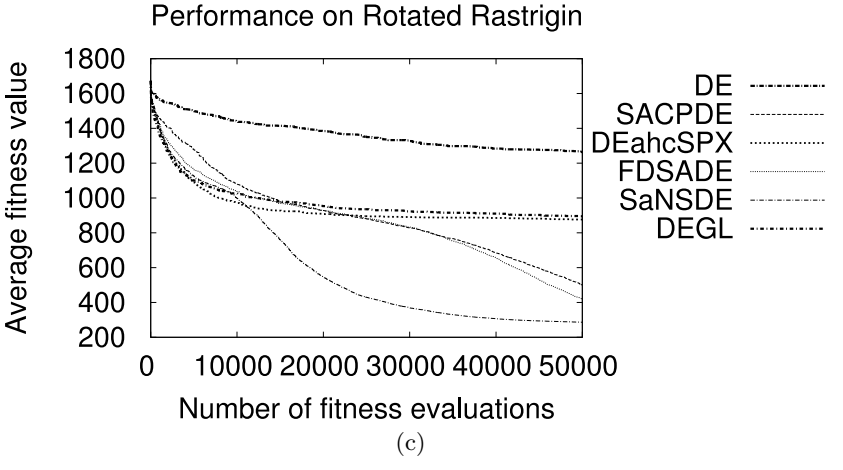
For the sake of completeness, some performance trends are shown in Figures 9.



(a)

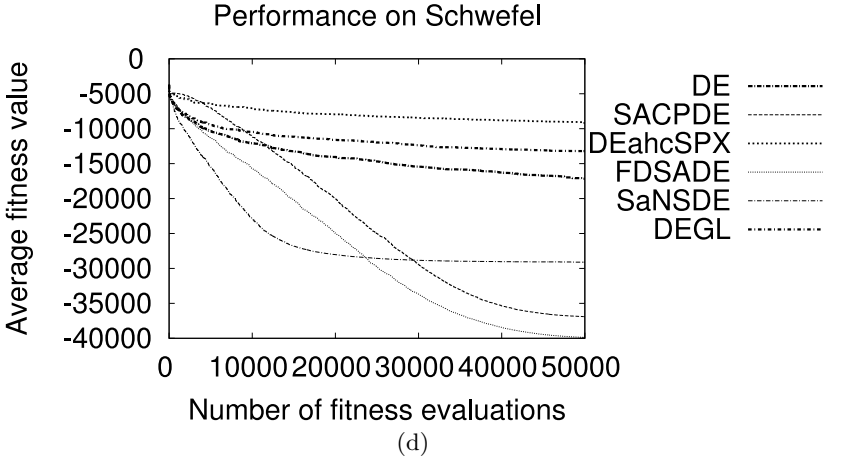


(b)

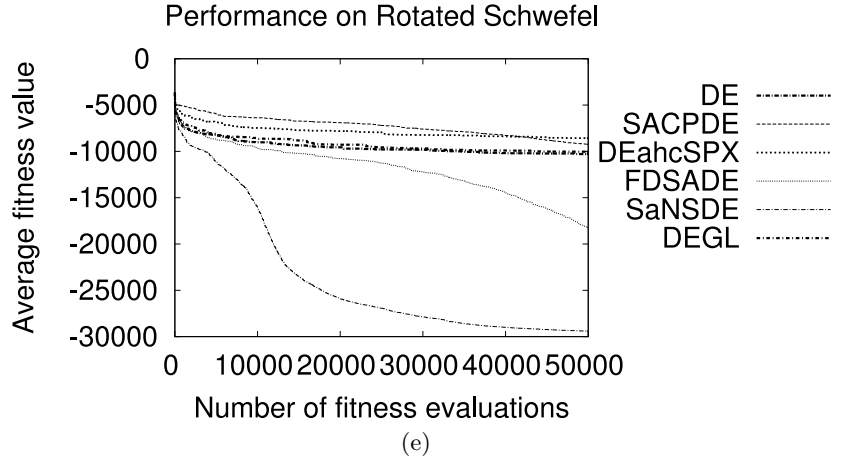


(c)

Fig. 9 Algorithmic performance over selected test problems in 100 dimensions



(d)



(e)

Fig. 9 (*continued*)

6 Conclusion

This chapter proposes integration of fitness diversity logic within the DE frameworks in order to self-adaptively affect the control parameter update and adaptively perform a dynamic population sizing. This integration is pursued by the definition of a novel index that measures the fitness diversity of a DE population and a novel algorithmic implementation.

The proposed algorithm has been tested on a set of twenty test problems and compared with a standard DE and four recently proposed DE based algorithms. Numerical results show that the proposed algorithm is very efficient in detecting solutions having high performance and that it has a robust behavior over a various set of test problems. The performance in convergence

speed is also competitive with those algorithms that represent the state-of-the-art in DE based implementation.

The fitness diversity logic is thus very efficient at monitoring the state of the algorithm during the optimization process and dynamically foreseeing algorithmic necessities. It is important to remark that the fitness diversity (self-)adaptation must aim at coordinating the exporative/exploitative pressure by increasing exploration when the diversity is low and exploitation when diversity is high. This mechanism tends to prevent the undesired effects of premature convergence and stagnation, therefore efficiently performing the optimization towards high quality solutions (as numerical results confirm). This aim is, in this chapter, pursued by frequently updating the scale factor and crossover rate and enlarging the population size in low diversity conditions; these operations give the algorithm a better chance of testing unexplored areas of the decision space and of, hopefully, detecting new promising solutions. Conversely, in high diversity conditions the population is shrunk and the control parameters kept constant; in this way the available genotypes and potential search directions (e.g., by means of the scale factor values) are exploited until the diversity decreases.

A remaining issue is that a proper fitness diversity index must be designed on the basis of the algorithmic structure (i.e., variation operators and selection mechanisms). The index proposed in this chapter seems to be very efficient in monitoring the state of the population of a DE, regardless of the optimization problem under examination. Therefore, we suggest its employment for designing (self-)adaptation within a DE framework.

References

1. Ali, M.M., Törn, A.: Population set based global optimization algorithms: Some modifications and numerical studies. Computers and Operations Research 31(10), 1703–1725 (2004)
2. Brest, J., Greiner, S., Boskovic, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10(6), 646–657 (2006)
3. Brest, J., Zumer, V., Maucec, M.: Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 215–222 (2006)
4. Caponio, A., Cascella, G.L., Neri, F., Salvatore, N., Sumner, M.: A fast adaptive memetic algorithm for on-line and off-line control design of PMSM drives. IEEE Transactions on System Man and Cybernetics-part B, special issue on Memetic Algorithms 37(1), 28–41 (2007)
5. Caponio, A., Neri, F., Tirronen, V.: Super-fit control adaptation in memetic differential evolution frameworks. Soft Computing-A Fusion of Foundations, Methodologies and Applications 13(8), 811–831 (2009)
6. Chakraborty, U.K. (ed.): Advances in Differential Evolution. Studies in Computational Intelligence, vol. 143. Springer, Heidelberg (2008)

7. Chakraborty, U.K., Das, S., Konar, A.: Differential evolution with local neighborhood. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 2042–2049 (2006)
8. Das, S., Abraham, A., Chakraborty, U.K., Konar, A.: Differential evolution with a neighborhood-based mutation operator. IEEE Transactions on Evolutionary Computation (to appear, 2009)
9. Deb, K.: Multi-objective Optimization using Evolutionary Algorithms, pp. 147–149. Wiley and Sons LTD, Chichester (2001)
10. Eiben, A.E., Schippers, C.A.: On evolutionary exploration and exploitation. Fundamenta Informaticae 35(1-4), 35–50 (1998)
11. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computation. Springer, Berlin (2003)
12. Feoktistov, V.: Differential Evolution in Search of Solutions, pp. 83–86. Springer, Heidelberg (2006)
13. Gämperle, R., Müller, S.D., Koumoutsakos, P.: A parameter study for differential evolution. In: NNA-FSFS-EC, WSEAS, pp. 293–298 (2002)
14. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Publishing Co., Reading (1989)
15. Krasnogor, N.: Toward robust memetic algorithms. In: Hart, W.E., Krasnogor, N., Smith, J.E. (eds.) Recent Advances in Memetic Algorithms. Studies in Fuzzines and Soft Computing, pp. 185–207. Springer, Berlin (2004)
16. Lampinen, J., Zelinka, I.: On stagnation of the differential evolution algorithm. In: Osmera, P. (ed.) Proceedings of 6th International Mendel Conference on Soft Computing, pp. 76–83 (2000)
17. Liu, J., Lampinen, J.: A fuzzy adaptive differential evolution algorithm. In: Proceedings of the 17th IEEE region 10 international conference on computer, communications, control and power engineering, vol. I, pp. 606–611 (2002)
18. Liu, J., Lampinen, J.: On setting the control parameter of the differential evolution algorithm. In: Proceedings of the 8th international Mendel conference on soft computing, pp. 11–18 (2002)
19. Liu, J., Lampinen, J.: A fuzzy adaptive differential evolution algorithm. Soft Computing - A Fusion of Foundations, Methodologies and Applications 9(6), 448–462 (2005)
20. Moscato, P.: On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. Rep. 790 (1989)
21. Neri, F., Mäkinen, R.A.E.: Hierarchical evolutionary algorithms and noise compensation via adaptation. In: Yang, S., Ong, Y.S., Jin, Y. (eds.) Evolutionary Computation in Dynamic and Uncertain Environments. Studies in Computational Intelligence, ch. 15, pp. 345–369. Springer, Heidelberg (2007)
22. Neri, F., Toivanen, J., Cascella, G.L., Ong, Y.S.: An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Transactions on Computational Biology and Bioinformatics, Special Issue on Computational Intelligence Approaches in Computational Biology and Bioinformatics 4(2), 264–278 (2007)
23. Neri, F., Toivanen, J., Mäkinen, R.A.E.: An adaptive evolutionary algorithm with intelligent mutation local searchers for designing multidrug therapies for HIV. Applied Intelligence, Special Issue on Computational Intelligence in Medicine and Biology 27(3), 219–235 (2007)

24. Neri, F., Kotilainen, N., Vapa, M.: A memetic-neural approach to discover resources in P2P networks. In: van Hemert, J., Cotta, C. (eds.) Recent Advances in Evolutionary Computation for Combinatorial Optimization. Studies in Computational Intelligence, ch. 8, pp. 119–136. Springer, Heidelberg (2008)
25. NIST/SEMATECH, e-Handbook of Statistical Methods (2003), http://www.itl.nist.gov/div898/handbook/
26. Noman, N., Iba, H.: Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation 12(1), 107–125 (2008)
27. Ong, Y.S., Keane, A.J.: Meta-lamarkian learning in memetic algorithms. IEEE Transactions on Evolutionary Computation 8(2), 99–110 (2004)
28. Price, K., Storn, R.: Differential evolution: A simple evolution strategy for fast optimization. Dr Dobb's J. Software Tools 22(4), 18–24 (1997)
29. Price, K.V., Storn, R., Lampinen, J.: Differential Evolution: A Practical Approach to Global Optimization. Springer, Heidelberg (2005)
30. Qin, A.K., Suganthan, P.N.: Self-adaptive differential evolution algorithm for numerical optimization. In: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 2, pp. 1785–1791 (2005)
31. Rahnamayan, S., Tizhoosh, H.R., Salama, M.M.: Opposition-based differential evolution. IEEE Transactions on Evolutionary Computation 12(1), 64–79 (2008)
32. Rechemberg, I.: Evolutionstrategie: Optimierung Technisher Systeme nach prinzipien des Biologishen Evolution. Fromman-Hozlboog Verlag (1973)
33. Storn, R., Price, K.: Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. Rep. TR-95-012, ICSI (1995)
34. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. Journal on Global Optimization 11, 341–359 (1997)
35. Sun, J., Zhang, Q., Tsang, E.: DE/EDA: A new evolutionary algorithm for global optimization. Information Science (169), 249–262 (2004)
36. Teo, J.: Exploring dynamic self-adaptive populations in differential evolution. Soft Computing–A Fusion of Foundations, Methodologies and Applications 10(8), 673–686 (2006)
37. Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., Rossi, T.: A memetic differential evolution in filter design for defect detection in paper production. In: Giacobini, M. (ed.) EvoWorkshops 2007. LNCS, vol. 4448, pp. 320–329. Springer, Heidelberg (2007)
38. Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., Rossi, T.: An enhanced memetic differential evolution in filter design for defect detection in paper production. Evolutionary Computation 16(4), 529–555 (2008)
39. Tsutsui, S., Yamamura, M., Higuchi, T.: Multi-parent recombination with simplex crossover in real coded genetic algorithms. In: Proceedings of the Genetic Evol. Comput. Conf (GECCO), pp. 657–664 (1999)
40. Wolpert, D., Macready, W.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1(1), 67–82 (1997)
41. Yang, Z., He, J., Yao, X.: Making a difference to differential evolution. In: Michalewicz, Z., Siarry, P. (eds.) Advances in Metaheuristics for Hard Optimization, pp. 397–414 (2008)

42. Yang, Z., Tang, K., Yao, X.: Self-adaptive differential evolution with neighborhood search. In: Proceedings of the World Congress on Computational Intelligence, pp. 1110–1116 (2008)
43. Zamuda, A., Brest, J., Boskovic, B., Zumer, V.: Differential evolution for multiobjective optimization with self adaptation. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 3617–3624 (2007)
44. Zielinski, K., Weitkemper, P., Laur, R., Kammeyer, K.D.: Parameter study for differential evolution using a power allocation problem including interference cancellation. In: Proceedings of the IEEE Congress on Evolutionary Computation, pp. 1857–1864 (2006)