JAISCR, 2012, Vol. 2, No. 4, pp. 279– 301

A STUDY ON THE SYNCHRONIZATION BEHAVIOUR OF DIFFERENTIAL EVOLUTION AND A SELF-ADAPTIVE EXTENSION

Valentino Santucci[1], Alfedo Milani[1,2], Flavio Vella[1]

[1] *Department of Mathematics and Computer Science, University of Perugia*
via Vanvitelli 1, 06123 Perugia, Italy

[2]*Department of Computer Science, Hong Kong Baptist University*
Kowloon Tong, Hong Kong, China
valentino.santucci@dmi.unipg.it, milani@unipg.it, flavio.vella@dmi.unipg.it

Abstract

Differential Evolution (DE) is a popular and efficient continuous optimization technique based on the principles of Darwinian evolution. Asynchronous Differential Evolution is a DE generalization that allows to regulate the synchronization mechanism of the algorithm by tuning two additional parameters. This paper, after providing a further experimental analysis of the impact of the DE synchronization scheme on the evolution, introduces three self-adaptive techniques to handle the synchronization parameters. Moreover the integration of these new regulatory synchronization techniques into state-of-the-art (self) adaptive DE schemes are also proposed. Experiments on widely accepted benchmark problems show that the new schemes are able to improve performances of the state-of-the-art (self) adaptive DEs by introducing the new synchronization parameters in the online automated tuning process.

1 Introduction

Optimization problems can be defined as the problems in which the objective is to maximize or minimize a given objective function. Recently, evolutionary and population-based algorithms [9] have gained increasing attention for their ability to locate the optimal (or a near-optimal) solution more quickly than traditional techniques, especially in the optimization of non-linear and non-differentiable functions. In this context Storn and Price [32, 22, 7, 5] proposed Differential Evolution (DE) which is a simple and powerful Evolutionary Algorithm (EA) for global optimization over continuous spaces. Its efficiency and robustness has been successfully demonstrated in various fields such as pattern recognition [16], mechanical engineering [24] and many applications of electrical engineering [28], as, for example, automated analog electronic circuits sizing [29].

In classical DE three control parameters are involved: the population size NP, the scale factor F, and the crossover probability CR. However, DE main peculiarity is its differential mutation operator that allows to self-adapt DE search at the landscape of the objective function at hand.

Classical DE, as other EAs, is characterized by a discrete generation model that synchronously evolves the population. Conversely, in [27] Qing proposed a sequential generation model where the individuals of the population are asynchronously and continuously updated.

In our previous work [19] Asynchronous Differential Evolution (ADE) has been introduced as a DE generalization that takes into account the synchronization mechanism of the algorithm allowing to DE a behaviour somewhat in the middle between the generational and the sequential model. This is achieved by introducing two additional parameters, i.e. the synchronization degree *SD* and a population shuffle strategy *SH*.

However, as well known, parameters tuning represents a major issue for every EA. Therefore, users are still faced with the problem of preliminary, or interactively, hand-tuning the algorithm parameters. This issue led researchers to consider techniques that automatically regulate DE parameters during the evolutionary process without the need of user interactions [41].

Moving from this consideration, this work, after providing a study of the impact of the new parameters on the evolutionary search of DE, introduces three self-adaptive schemes for *SD* and *SH* thus providing a viable way towards an efficient and definitive parameters-free DE. Furthermore, these synchronization regulatory techniques have been integrated in two of the most representative (self) adaptive DE proposals (SaDE [26] and jDE-2 [3]) thus resulting into six completely adaptive DE schemes.

The rest of the paper is organized as follows. An overview of classical DE is shown in Section 2. State-of-the-art of the works related to DE synchronization model and DE adaptive schemes are provided in Sections 3 and 4. The synchronization degree and shuffle strategies introduced in [19] are recalled in Section 5, while the self-adaptive schemes of these new parameters are introduced in Section 6. Experimental results aiming at point out the impact of the synchronization degree on the evolution and the improvements obtained with the new adaptive DE schemes are reported in Section 7. Finally, Section 8 concludes the paper by introducing also some possible future lines of research.

2 The Differential Evolution algorithm

Storn and Price [32] proposed Differential Evolution (DE) which is a simple and powerful technique for optimizing non-linear and even non-differentiable real functions. Substantially, DE is a population-based evolutionary algorithm (EA) that exploits a population of potential solutions in order to probe the search space.

DE presents some similarities with traditional EAs, indeed, it uses the genetical operators of mutation, crossover, and selection as in Genetic Algorithms (GAs) [18]. However, unlike the original binary GAs, it does not employ a binary encoding but uses a population of NP D-dimensional real values vectors.

DE initially generates a random population of candidate solutions uniformly distributed in the solutions space. At each generation G, DE performs the mutation and the crossover operations to produce a *trial* vector for each individual, also called *target* vector, in the current population. Each target vector is then replaced in the next generation by the associated trial vector if and only if the produced trial presents a better fitness than the target. This process is iteratively repeated, through the so called generations, until a stop criterion is met (e.g. a given amount of fitness evaluations has been performed).

In the following subsections the three main DE phases (mutation, crossover, and selection) are briefly described.

2.1 Mutation

The mutation phase, for each target individual $x_{i,G}$, generates a *mutant* vector $v_{i,G+1}$. The basic mutation scheme (”rand/1”) originally proposed in [32] computes the mutant as follows:

$$v_{i,G+1} = x_{r_0,G} + F \cdot (x_{r_1,G} - x_{r_2,G}) \qquad (1)$$

where r_0, r_1, r_2 are three random integers in $[1, NP]$ mutually different among them. x_{r_0} is called *base vector*, while $x_{r_1} - x_{r_2}$ is the *difference vector*, and $F > 0$ is the *scale factor* parameter.

This differential mutation operator represents the main peculiarity of DE. Indeed, it confers to the algorithm the ability to automatically adapt the mutation step size and orientation to the fitness landscape of the given optimization problem, other than to shade the DE search from an explorative to an exploitative behaviour with the passing of generations.

In [24] and [12] other mutation schemes have been proposed and generalized. In the following we report the two most popular ones, i.e. "best/1" and "current-to-best/1", that differ from (1) for a different choice of the base vector:

$$v_{i,G+1} = x_{best,G} + F \cdot (x_{r_1,G} - x_{r_2,G}) \tag{2}$$

$$v_{i,G+1} = x_{i,G} + F \cdot (x_{best,G} - x_{i,G}) + F \cdot (x_{r_1,G} - x_{r_2,G}) \tag{3}$$

where r_1 and r_2 are random integers in $[1, NP]$ mutually different among them, while $x_{best,G}$ is the best fit individual until generation G.

Since the use of the best fit individual in the base vector part of the differential mutation operator, these two latter schemes, conversely from the former one, shift the DE exploration/exploitation balance towards exploitation.

2.2 Crossover

After mutation, a binomial crossover operator generates a population of NP trial vectors, i.e. $u_{i,G+1}$, by recombining each pair composed by the generated mutant $v_{i,G+1}$ and its corresponding target $x_{i,G}$. Indeed, each trial vector is formed by taking some components from the target vector and some other ones from the mutant, according to the crossover probability $CR \in [0,1]$.

Formally, the j-th component of the trial vector $u_{i,G+1}$ is computed according to:

$$u_{i,G+1,j} = \begin{cases} v_{i,G+1,j} & \text{if } \theta_{1,j} \leq CR \vee \theta_{2,j} = j \\ x_{i,G,j} & \text{otherwise} \end{cases} \tag{4}$$

where $\theta_{1,j} \in [0,1]$ is a random number generated for each dimension j and $\theta_{2,j} \in [1, D]$ is a random integer generated for each trial vector which ensures that $u_{i,G+1}$ inherits at least one component from the mutant $v_{i,G+1}$.

This crossover scheme is usually denoted as *binomial crossover* and, although it is not the only one proposed (see [24] for a comprehensive review), it is generally assumed as the standard DE crossover scheme.

2.3 Selection

After crossover phase there are two populations with a one-to-one correspondence: the target individuals $\{x_{i,G}\}$ and the trials $\{u_{i,G+1}\}$. Therefore, in the selection phase, the next generation population is selected by a one-to-one tournament among each target and its associated trial, thus the best fit individual of the two enters in the next generation. More formally, in the case of a minimization problem:

$$x_{i,G+1} = \begin{cases} u_{i,G+1} & \text{if } f(u_{i,G+1}) \leq f(x_{i,G}) \\ x_{i,G} & \text{otherwise} \end{cases} \tag{5}$$

where $f : [a,b]^D \to \mathbb{R}$ is the objective/fitness function to minimize.

3 Synchronization Schemes in DE

The evolution mechanism of classical Differential Evolution (DE) relies on a population updating procedure that follows the so called "discrete generation model" or "generational model" for short. In this scheme, during a generic generation G, the whole population remains unchanged until it is completely replaced at the begin of the next generation $G+1$. Therefore, in generation G the DE operations of mutation and crossover generate a temporary population of trial vectors using the current population produced at the end of the previous generation $G-1$, then the one-to-one selection among temporary and current population replaces every individuals in the next generation $G+1$. Using this generational model we can say that generations act as synchronization points thus DE population is synchronized every NP fitness evaluations (performed in the single generation).

However, the previous described generational model is not the only one applicable to DE. A completely asynchronous population updating procedure is adopted, for example, in [27] and [37] and it follows the usually called "continuous generation model" or "sequential model". In this case all the three DE operations of mutation, crossover, and selection are sequentially performed on every individual, thus the new born offspring immediately enters the population of candidate solutions without waiting until the synchronization point. Therefore the evolution of each individual is performed using a population updated with last "evolutionary results".

From the point of view of memory requirements, the sequential (or completely asynchronous) model allows to use only one population while generational DE requires a double sized memory in or-

der to store current and trial populations. Moreover, the sequential model, thanks to the use of a "freshly" updated population, generally presents a faster convergence speed, however this may lead to entrapment in a local optimum thus reducing the algorithm robustness with respect to generational DE. These behaviours are due to the different diversity rates obtained by the two model. As can be seen in the section related to experiments (Section 7), in sequential DE the diversity of the individuals in the population is shrinking at a grater rate than classical DE. This is also reasonable since the population of classical DE remains stable during the duration of any single generation, while sequential DE continuously diversifies the population every fitness evaluation thus it is very plausible that every NP fitness evaluations (i.e. one generation in classical DE) the asynchronous scheme is able to reach a smaller degree of diversity with respect to the generational updating scheme. Finally, it is important to note that this variable velocity of the diversity rate highly impacts on the exploration/exploitation balance of DE search process.

Moreover, in completely asynchronous DE a newborn trial vector can be naturally compared with an arbitrary individual of the population, so different selection schemes can be adopted. Tagawa in [34] introduces the random and worst survival selection methods that select for comparison, respectively, a random or the worst fit individual of the population. Another important selection scheme that can not be directly implemented in generational DE is the crowding selection [30] that replaces the population individual genotypically closer to the considered trial.

Population updating procedure lying somewhat in the middle between completely asynchronous and classical DE have also been proposed. Transversal DE (TDE) [12] and Dispersive DE (DDE) [35] extend the sequential scheme by introducing a new parameter TS and allowing an individual to evolve TS times before replacing the target one in the population. While in TDE the population is updated every "transversal step", in DDE the population update is performed only after all the TS transversal steps have been performed. Although Feoktistov in [12] claims that TDE allows intermediate behaviours between sequential and classical DE, it can be noted that, while the sequential DE can be easily reproduced by setting $TS = 1$, the generational DE can not be reproduced with any value of the TS parameter.

The necessity to have a population updating procedure that naturally shades from a completely asynchronous to a synchronous behaviour has led us to propose Asynchronous DE (ADE) [19]. ADE introduces a new synchronization degree parameter SD that is able to naturally reproduce sequential DE when $SD = 1$ and classical DE when $SD = NP$. Moreover, as described in Section 5, a "super-synchronous" behaviour is also implemented by setting $SD > NP$ and also different population ordering criteria have been considered thus allowing DE to approximate the random and worst survivor selection methods previously described also in the case of a not completely asynchronous update scheme. ADE, substantially, implements in DE context what is the Generation Gap concept proposed for GAs in [8, 10].

However, although different parameters that regulates the DE population updating procedure (i.e. SD in ADE and TS in TDE/DDE) have been proposed, there is no proposal for an adaptation scheme that automatically regulates these parameters during the evolution thus relieving the user from the issue of hand-tuning them. Moving from this consideration, in this paper, we introduce a self-adaptive scheme for the synchronization degree parameter (SD) of ADE.

4 Parameters Adaptivity in DE

Differential Evolution (DE) control parameters need to be adjusted in order to adapt DE search at the given problem. As described in Section 2, these parameters are the values NP, F, CR in addition to the non-numerical parameters which regulate the mutation and crossover strategies [24].

Concerning the three numerical parameters, Storn and Price [32] suggest: $NP = 10 \cdot D$ (where D is the problem dimensionality), $F \in [0.5, 1]$, $CR \in [0.8, 1]$. Many other works have discussed the parameters role and tuning in DE [21, 40, 38]. The commonly recognized idea is that CR is more sensitive to problem properties such as modality and separability (low CR values are a good choice for separable problems and values close to 1 are pre-

ferred for non-separable objective functions), while F is more related to convergence speed. Instead, NP is generally regulated in order to handle different problem dimensionalities. Binomial crossover (rule (4)) is usually preferred to other crossover schemes, while the two most applied mutation operators are "rand" and "current-to-best" (defined respectively in rules (1) and (3)). Using the notation originally proposed in [32], the DE schemes that use binomial crossover and the two mutation strategies aforementioned are called, respectively, "rand/1/bin" and "current-to-best/1/bin".

The theoretical and experimental studies conducted [14, 38, 17] show that different DE control parameters are quite dependent on characteristics of the given optimization problem. The main reasons are that different parameters settings lead to different exploration/exploitation balances [11] by also affecting the population diversity shrinking rate [39] which in turn heavily affect DE performances on the problem at hand leading in the worst case to a slow convergence or to entrapment in local optima. Therefore, a single parameters setting can work well for a particular class of problems and be completely unsuitable for another class, furthermore, it is reasonable to think that different parameters settings might be preferred in different evolution stages.

Hence, although DE has been shown to be a simple and powerful optimization technique, users are still faced with the problem of preliminary, or interactively (at different stages), hand-tuning the algorithm parameters. This tuning process may require a huge amount of computational time other than a deep insight about the problem by the user. These issues led researchers to consider techniques that automatically regulate DE parameters during the evolutionary process without the need of user interactions.

Different adaptive schemes of DE have been proposed [41]. They can be primarily classified in two categories: the ones that use a deterministic logic to modify parameters values, and the ones based on an adaptive or self-adaptive mechanism that exploits some feedback from evolutionary process. For the first category it can be mentioned DETVSF [6] that linearly shades F from 1 to 0.5 with the passing of generations, in this way DE should move from an explorative to an exploitative behaviour during the evolution. However more recent proposals belong to the second category and, thanks to the use of an evolution feedback, they result more performant than the deterministic schemes.

An aspect common to all the adaptive or self-adaptive DEs is the general scheme that works by dividing the evolutionary process into multiple temporal phases and performing a parameters learning procedure in each phase in order to hopefully decide the more suitable parameters setting for the next phase. The decision mechanism is generally based on current DE performances such as the number of trial vectors successfully entering in DE population or the fitness gain.

It is also worthwhile to note that in [24] two new terms are defined: *dither* and *jitter*. *Dither* identifies the practice of using a different parameter value (e.g. F or CR) for each individual of the population, while *jitter* refers to the use of a different parameter value (generally only for F) for each problem dimension other than for each individual. Anyway, in (self) adaptive schemes it is commonly adopted the *dither* approach, also because the weights of the problem dimensions are implicitly and automatically adapted by the differential mutation operator of DE.

In the following a brief description of six (self) adaptive DE schemes is provided.

4.1 FADE

Fuzzy Adaptive Differential Evolution (FADE) has been introduced by Liu and Lampinen [15]. It adapts the control parameters F and CR by means of a fuzzy controller [23]. Basically, it is composed by a list of hand-tuned fuzzy "if-then" rules which regulate the DE parameters in different ways. The rules are selected and applied using objective function feedback as control input.

4.2 DESAP

Differential Evolution with Self-Adapting Populations (DESAP) has been introduced by Teo [36]. DESAP adapts all the DE numerical parameters, i.e. the population size NP other than F and CR.

Each population individual is extended with a local version of the three parameters NP, F, and

CR. While F and CR are adapted using a differential mutation scheme similar to the one used for individual mutation, the DESAP main idea is the adaptation of the population size NP.

Two different DESAP schemes are proposed: DESAP-Abs, and DESAP-Rel. The initial NP is set to $10 \cdot D$ (where D is the problem dimensionality). In both schemes the individuals attributes relative to population size are stochastically evolved by Gaussian perturbation and differential mutation. At the end of a learning stage the average of the population size attributes is computed and used as the new NP for subsequent generations. If the new population has a greater size than the previous one then the worst fit individuals of the previous population are discarded, in the other case the previous best fit individuals are cloned until new population size is reached. DESAP-Abs and DESAP-Rel differ for the use of, respectively, absolute values or growing rates as population size attributes.

4.3 SaDE

Self-adaptive Differential Evolution (SaDE) has been proposed by Qin and Suganthan in [26] and extended in [25]. It automatically adapts the control parameters F and CR other than the learning strategy.

For each target individual, F is allowed to vary in $(0,2]$ and is randomly sampled from a Gaussian distribution, with mean 0.5 and standard deviation 0.3, denoted by $N(0.5,0.3)$. Instead, CR is randomly sampled, for each target individual, from the Gaussian distribution $N(CR_m,0.1)$. CR_m is initialized at 0.5 and updated every 25 generations. A different CR value is generated for each individual that maintains it for 5 generations, in this way, after 25 iterations, every individual has changed its CR exactly 5 times. At the end of this learning stage the new CR_m is computed by averaging over all the CR values that have led to the selection of a trial vector for the next generation.

Concerning DE learning strategy, the first SaDE proposal [26] adopts two different strategies while the second version [25] extends the first one by employing two more strategies for a total of four strategies. However, in the following we describe the first SaDE since it is the one referred in Sections 4.4 and 6. Hence, the two learning strategies employed are "rand/1/bin" and "current-to-best/1/bin" that, for the convenience of the following description, we call, respectively, strategy 1 and strategy 2.

The probability of applying strategy 1 to each individual is p_1, while $p_2 = 1 - p_1$ is the probability of applying strategy 2. They are initialized at $p_1 = p_2 = 0.5$. These values are modified every learning stage composed by a given number of generations (50 is used in [26] and in our experiments). During this stage, the number of successes and failures of the produced trials are recorded, respectively, in ns_1, nf_1 for strategy 1, and ns_2, nf_2 for strategy 2. Then the probabilities are updated according to:

$$\begin{aligned} p_1 &= \frac{ns_1 \cdot (ns_2 + nf_2)}{ns_2 \cdot (ns_1 + nf_1) + ns_1 \cdot (ns_2 + nf_2)} \\ p_2 &= 1 - p_1 \end{aligned} \tag{6}$$

Since the successes/failures counters are resetted every learning stage, the new probabilities reflect the strategies success rates obtained in the last generations avoiding possible side effects accumulated in previous stages.

Finally, in order to speed up the convergence, SaDE applies a local search procedure (the Quasi-Newton method) on some selected solutions every 200 generations.

4.4 jDE-2

Brest and others proposed jDE-2 in [3] as an improvement of their previous self-adaptive DE [4]. The proposed technique adapts the numerical parameters F and CR other than the DE learning strategy. In this latter case the same scheme of SaDE [26] is adopted (see Section 4.3).

Each individual is extended with two couples of the control parameters F and CR, one for each strategy. More formally, individual i is associated to F_i^1, CR_i^1, F_i^2, CR_i^2. Therefore, the trial associated with individual i is generated using the couple of parameters F_i^s and CR_i^s, where s is the selected learning strategy (that is 1 or 2).

The numerical parameters are initialized to $F_i^1 = F_i^2 = 0.5$ and $CR_i^1 = CR_i^2 = 0.9$, then new parameters values are computed before differential mutation is performed and replace the old ones if and only if the produced trial presents a better fitness than the target solution with which is com-

pared. In this way the better values of these control parameters lead to better individuals which, in turn, are more likely to survive and produce offspring and, therefore, propagate these parameters values. The computation of the new control parameters follows the following rules:

$$F^s_{i,G+1} = \begin{cases} 0.1 + r_1 \cdot 0.9 & \text{if } r_2 < 0.1 \\ F^s_{i,G} & \text{otherwise} \end{cases} \quad (7)$$

$$CR^s_{i,G+1} = \begin{cases} r_3 & \text{if } r_4 < 0.1 \\ CR^s_{i,G} & \text{otherwise} \end{cases} \quad (8)$$

where r_1, r_2, r_3, r_4 are uniform random numbers in $[0,1]$ and $F^s_{i,G}$ is the F value for strategy s of individual i at generation G (a similar meaning is present in the CR update rule (8)).

Out-of-bounds trial components are avoided using a technique that mixes the "reflecting back" [31] and "setting on bounds" [24] methods:

$$u_{i,j,G} = \begin{cases} a_j & \text{if } t \leq 0.5 \wedge u_{i,j,G} < a_j \\ b_j & \text{if } t \leq 0.5 \wedge u_{i,j,G} > b_j \\ 2 \cdot a_j - u_{i,j,G} & \text{if } t > 0.5 \wedge u_{i,j,G} < a_j \\ 2 \cdot b_j - u_{i,j,G} & \text{if } t > 0.5 \wedge u_{i,j,G} > b_j \end{cases} \quad (9)$$

where a_j and b_j are, respectively, the given lower and upper bounds and t is a uniform random number in $[0,1]$.

Finally, in order to avoid a premature convergence, every l generations, jDE-2 replaces the k worst individuals with new randomly selected ones, but without evaluating them. In [3] and in our experiments the values for l and k are set to $k = 30$ and $l = 100$.

4.5 JADE

JADE has been recently proposed by Zhang and Sanderson [42]. It implements the new mutation strategy "current-to-p-best" and controls F and CR by using the two self-adaptive parameters μ_F and μ_{CR}.

The mutation operator introduced is a generalization of the "current-to-best" strategy (rule (3)). It attempts to diversify the population without losing the fast convergence property of "current-to-best" at the cost of maintaining a fitness-based rank of the population. Indeed, "current-to-p-best" strategy generates a mutant vector according to:

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

where $x^p_{best,G}$ is uniformly chosen as one of the top $100p\%$ individuals in the current population (therefore $p \in (0,1]$).

At each generation the crossover probability CR_i of individual i is generated by sampling a Gaussian distribution with mean μ_{CR} and standard deviation 0.1, while μ_{CR} is updated at the end of each generation according to:

$$\mu_{CR} = (1-c) \cdot \mu_{CR} + c \cdot (S_{CR}) \quad (11)$$

where S_{CR} is the set of successful crossover probabilities, c is a control parameter, and $(\cdot)$ is the usual arithmetic mean.

Instead, the scale factor F_i of individual i is sampled from a Cauchy distribution with location parameter μ_F and scale parameter 0.1. μ_F is updated at the end of each generation according to:

$$\mu_F = (1-c)\mu_F + c \cdot (S_F) \quad (12)$$

where S_F is the set of successful scale factors, c is the same control parameter of (11), and $(\cdot)$ is the Lehmer quadratic mean:

$$(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F} \quad (13)$$

Note finally that in [42] values for the parameters c and p are suggested.

4.6 DEPD

DEPD has been proposed by Ali and Törn in [1]. Various modifications to classical DE have been made like the use of two evolving populations. Crossover rate CR has been fixed to 0.5, while the main innovation concerns the scale factor F that, conversely from the previous described schemes, is regulated by means of a fitness-based adaptation mechanism:

$$F = \begin{cases} \max\{l_{min}, 1 - |\frac{f_{max}}{f_{min}}|\} & \text{if } |\frac{f_{max}}{f_{min}}| < 1 \\ \max\{l_{min}, 1 - |\frac{f_{min}}{f_{max}}|\} & \text{otherwise} \end{cases} \quad (14)$$

where $l_{min} = 0.4$ is a lower bound for F, while f_{min} and f_{max} are the minimum and maximum fitness values over the population individuals obtained in the last generation.

5 Asynchronous Differential Evolution

Asynchronous Differential Evolution (ADE) is a generalization of classical DE that takes into account the synchronization mechanism of the algorithm. ADE has been introduced in [19] where its potential benefits for DE performances have been shown.

As described in Section 2, classical DE is characterized by a generational loop composed by the phases of mutation, crossover, and selection. In this loop it is generally assumed a hidden and implicit synchronization mechanism ensuring that crossover and selection phases are performed exactly with NP couples of individuals of the form target/trial. This is due to the fact that the mutation phase generates exactly NP mutants.

Algorithm 1 ADE

```
1:  procedure ADE
2:      X.length ← NP
3:      V.length ← SD
4:      INITIALIZE(X)
5:      from ← 1
6:      while termination criterion is not met do
7:          V ← MUTATION(X)
8:          to ← from + min{SD, NP} − 1
9:          if to > NP then
10:             to ← NP
11:         end if
12:         X[from:to] ← CR_SEL(V, X[from:to])
13:         UPDATE_GLOBAL_BEST()
14:         from ← to + 1
15:         if from > NP then
16:             from ← 1
17:             SHUFFLE(X)
18:         end if
19:     end while
20: end procedure
```

ADE made explicit the DE synchronization mechanism by introducing a parameter SD that regulates the number of mutant vectors produced by mutation phase. Since SD can naturally assume positive integer values from 1 to NP, DE is now able to vary from a completely asynchronous behaviour, i.e. $SD = 1$, to a completely synchronous one, i.e. $SD = NP$ (the classical DE). Moreover, a further extension has been proposed by allowing SD values greater than NP in order to model a "super-synchronous" DE. In this latter case selection is performed among more than two individuals.

ADE is formally described by the pseudo-code reported in Algorithm 1. X and V represent, respectively, the set of the NP population individuals and the set of the SD mutants. Instead, the notation $X[a : b]$ is used to represent the subset $\{X_a, X_{a+1}, \ldots, X_b\} \subseteq X$.

MUTATION and *CR_SEL* implement the three DE phases. The *MUTATION* routine, given the DE population in input, produces the mutants according to one of the DE mutation strategies (see Section 2.1), while the *CR_SEL* routine modifies the block of population in input (even all the population if $SD \geq NP$) using the mutants previously produced, i.e. it performs crossover and selection according to (4) and (5). With this approach, unlike in classical DE, the order of the individuals is generally relevant. For this reason a population shuffling procedure, i.e. *SHUFFLE*, is performed, according to some criterion, every time that every one of the NP individuals has been evolved. Finally, at each iteration the global best value is updated. Although this seems to be a marginal aspect, note that this is not true, indeed, in the case of a mutation scheme employing the best fit individual (like (2) or (3)) the responsiveness of the algorithm to improvements becomes more immediate.

As a result of this generalization, the concept of generation is now decoupled from that of main-loop iteration. Indeed, since in each loop iteration only SD individuals are evolved, a complete population renewal requires generally more than one iteration. As aforementioned, when $SD = NP$ the classical DE behaviour is reproduced. Instead, when the $SD < NP$ a certain degree of asynchrony is introduced. As consequence of the fact that a generation persists for more than one main-loop iteration, the evolution proceeds by using "freshly updated" solutions thus resulting in a more greedy and exploitative search. Obviously, as SD approaches NP the greediness decreases and potential population diversity increases.

In the asynchronous case, i.e. $SD < NP$, the exact number of main-loop iterations needed for a completely population renewal is $m = \lceil NP/SD \rceil$. This means that the population is split in m chunks evolved separately one after the other. Individuals in the first chunks are more likely to be associated to mutants formed by using the previous generation population, while individuals in the last chunks tend

to be associated to mutants produced by freshly updated individuals (due to the previous chunks evolution). This mechanism confers a more greedy and exploitative evolution to last chunks and is the reason of why when $SD < NP$, unlike the classical case $SD = NP$, the order of individuals is not irrelevant.

In the super-synchronous case, i.e. $SD > NP$, in each population update loop, more than NP fitness evaluations are performed resulting in a more accurate navigation of individual neighborhood, especially in later loops when the step size becomes small. Mutation routine produces a mutants population V containing more vectors than the individuals population X, thus that at least, and not exactly, one mutant is associated to a target individual. Indeed, the mutant vectors associated to x_i have the form $v_{i+k\cdot NP}$ where k is a non-negative integer, i.e. mutants are assigned to target vectors in a circular way until the end of mutants population is reached. In crossover phase the mutants assigned to the same target vector are combined with it and an equal number of trial vectors $u_{i+k\cdot NP}$ is generated. Then the selection is performed among the trials and the target individuals resulting in a tournament among, usually, more than two vectors. For example, in the case $SD = 2 \cdot NP$, the next generation vector is computed according to $x_{i,G+1} = \arg\min\{f(x_{i,G}), f(u_{i,G+1}), f(u_{i+NP,G+1})\}$. When SD is a multiple of NP, each target is associated to the same number of mutant vectors, i.e. SD/NP, but this is no more true if SD is not a multiple of NP. In this latter case, first individuals are associated to one more vector than the last ones thus that also in this case the order of individuals in the population can not be considered as irrelevant.

It is worthwhile to note that both the asynchronous and super-synchronous case naturally "converge" to classical DE when $SD = NP$ and that when $SD = 1$ sequential DE is reproduced. Instead, the super-synchronous case is related to classical DE in a similar way of how Dispersive DE [35] and Transversal DE [12] are related to sequential DE.

In [19], three different shuffle strategies have been proposed:

- *static*: it is the implicit ordering criterion adopted in classical DE, i.e. it uses for all the generations the same order established at the startup (complexity $O(1)$),
- *dynamic*: consists in computing a random permutation of the individuals at the end of every generation (complexity $O(NP)$ using Fisher-Yates method [13]),
- *best*: at the end of every generation the individuals are sorted from the best fit to the worst fit (complexity $O(NP \cdot \log NP)$).

The classical *static* criterion has been reported only for completeness, since, although its low complexity, with a not so huge population size it has no reason to be adopted. Indeed, in the classical case $SD = NP$, the use of a *dynamic* shuffle instead of a *static* one does not impact on the search scheme of the algorithm[1]. In [19] an analysis of the possible couplings between shuffle and mutation strategies have been provided. The experimental results suggest to couple: the *dynamic* shuffle strategy with a purely random mutation scheme (like (1)) and the *best* shuffle criterion with a mutation scheme that involves the population best individual (like (2) or (3)). Finally, it is interesting to note that the different ordering criteria here proposed are able to approximate the survival selection schemes introduced in [34].

Summarizing, the new parameters introduced by ADE are the synchronization degree SD and the shuffle strategy that will be referred as SH in the following sections.

6 Self-adaptive Asynchronous DE

In this paragraph we introduce some Self-adaptive DE schemes that take into account the synchrony regulatory parameters proposed in [19].

In [19] and in the more recent experiments discussed in Section 7 of this paper it has been empirically proved that different synchronization behaviours are able to improve DE performances by acting on the population diversity shrinking rate. Moreover, as described in Section 4, a parameters self-regulatory mechanism allows DE to automatically adapt its search at the fitness landscape of the given optimization problem without the need of an

[1] This change has the same effect of changing the order of the outcomes of the random number generator, therefore, since the outcomes are independent among them, there is no difference in applying *static* or *dynamic* shuffling to classical DE.

expensive parameters tuning by the user. Moving from these considerations, a natural choice is to introduce the synchronization parameters *SD* and *SH* in (self) adaptive DE schemes.

After analyzing and evaluating the state-of-the-art (self) adaptive DEs (see Section 4) we propose new (self) adaptive asynchronous (SAA) schemes on top of jDE-2 [3] and of a variant of SaDE [26] referred in the following with the acronym "SaDE*". The choice of these two base algorithms is due to the fact that they are the state-of-the-art most representative schemes of the categories of adaptive DEs (SaDE) and self-adaptive DEs (jDE-2)[2].

SaDE has been modified in SaDE* aligning it with jDE-2 in order to obtain a more significant comparison. Conversely from SaDE, in SaDE* no local search procedure is employed, however, the population partial replacement procedure of jDE-2 (described in Section 4.4) is adopted. It is also important to note that SaDE* and jDE-2 natively implements the same learning strategy adaptation scheme.

Furthermore, both in SaDE* that in jDE-2, an improvement in the initialization phase has been introduced. Half population is generated using a completely random method according to (15), while the other half is initialized in a random bound of the bounding box that defines the feasible search space according to (16). Note that this initialization scheme is applied not only at the start of the algorithm but also every time that the worst individuals replacement procedure is invoked, i.e. every 100 generations.

$$x_{i,j} = a_j + r \cdot (b_j - a_j) \tag{15}$$

$$x_{i,j} = \begin{cases} a_j & \text{if } r \leq 0.5 \\ b_j & \text{otherwise} \end{cases} \tag{16}$$

where r is a random number in $[0,1]$ and a_j, b_j are the lower and upper bounds for dimension j. In this way, the algorithm becomes more efficient for those problems having the global optimum located on the the search space bounds.

In order to adapt the synchronization parameters *SD* and *SH*, note that, in ADE, the concept of generation is decoupled from that of main-loop iteration (see Section 5). Indeed, since a generation is generally composed by more than one iteration, in the following we use the word "generation" to indicate a completely renewal of the population, i.e. a generation ends after that *NP* new trials have been generated (or, equivalently, *NP* fitness evaluations have been performed).

Basing on experimental results (see [19] and Section 7), the allowable domains for parameters *SD* and *SH* have been restricted. Since a super-synchronous DE is rarely effective and the classical static shuffle strategy has no meaning to be adopted in a not synchronous behaviour, we have decided to restrict the values for *SD* in $[1, NP] \cap \mathbb{N}$ and to allow only the *dynamic* and the *best* ordering criteria for *SH*. Hence, for the convenience of the following description we use $SH = 1$ and $SH = 2$ to indicate, respectively, *dynamic* and *best* shuffle strategies.

Since the adaptation mechanism used in SaDE* and jDE-2 for *F* and *CR* works at individual level, it could not be used for *SD* and *SH*. Indeed, these two parameters do not refer to a single individual but to the whole population. For this reason we have decided to split the evolution in different learning stages, each one composed by a given number of generations. Different *SD* values and *SH* strategies are adopted for different generations of the same stage, thus the performances of the *SD* and *SH* choices are recorded and used to adjust these parameters for the next evolution stage.

For the shuffle strategy *SH*, a simple mechanism that exploits the learning strategy adaptation, common to SaDE* and jDE-2, has been adopted. From experiments on ADE [19], it is emerged that the best couplings for what regards learning and shuffle strategies are "rand/1/bin"-"dynamic" and "best/1/bin"-"best". Unfortunately, in our base algorithms, instead of the learning strategy "best/1/bin" it is employed "current-to-best/1/bin". However, since both these strategies exploit the population best individual, and comforted also by some preliminary experiments, we have concluded that the "best" shuffle method is suitable also for "current-to-best/1/bin". Therefore, at each generation, we apply the shuffle strategy $SH = s$ basing on the same probability p_s (where s is 1 or 2) learned every 50 generations for the DE learning strategy (see Subsection 4.3). In this way the suggested ideal couplings have more probabilities to be applied, although other couplings are not avoided.

[2] For a detailed discussion about the difference between adaptive and self-adaptive DEs refer to [4].

Instead, for the synchronization degree parameter *SD*, three different adaptation schemes have been introduced. The first scheme SAA1 makes use of purely probabilistic choices, the second one SAA2 tries to variate some *SD* values in the direction learned from previous evolution stage, while SAA3 employs a purely deterministic update of *SD* that is not based on the evolution feedback. These three schemes are detailed in the following subsections.

6.1 SAA1: the probabilistic strategy

In SAA1 five equally spaced synchronization degrees are involved, i.e.: $SD_1 = 1$, $SD_2 = NP/4$, $SD_3 = NP/2$, $SD_4 = 3 \cdot NP/4$, $SD_5 = NP$. Each SD_k is associated to its probability to be applied q_k. Therefore, at each generation one of SD_k synchronization degrees is employed according to the probability distribution $\{q_k\}$. During the evolution the probabilities q_k are updated, while the SD_k values remain unchanged throughout the entire execution.

The probabilities are equally initialized to $q_k = 0.2$ for each k, while they are updated at the end of each learning stage of the duration of 50 generations. In each learning stage, the number of trials successfully entering into next generation while the SD_k value is employed, are recorded as $succ_k$, instead, the amount of discarded ones is recorded in $fail_k$. Therefore the probabilities for the next stage are updated according to $q_k = succ_k/(succ_k + fail_k)$. Then they are normalized ($q_k = q_k/\sum_{i=1}^{5} q_i$ for each k) in order to sum up to 1.

Side-effects due to previous stages are avoided by resetting the counters $succ_k$, $fail_k$ after every update. Finally, note that, to prevent possible divisions by zero, in the first 5 generations of each learning stage all the SD_k are employed, thus ensuring that each one is used at least one time.

6.2 SAA2: the "moving values" strategy

In SAA2, as in SAA1, five synchronization degrees, initialized to $SD_1 = 1$, $SD_2 = NP/4$, $SD_3 = NP/2$, $SD_4 = 3 \cdot NP/4$, $SD_5 = NP$, have been adopted. However, conversely from the previous strategy, the SD_k values are not associated to probabilities and are not anchored to their initial values during the entire evolution.

Every 5 generations all the SD_k are employed exactly one time. However, in order to avoid possible biases due to their order, they are randomly permutated every 5 generations. As in SAA1, $succ_k$ records the number of trials promoted to next generation while SD_k is employed. This value is accumulated within a learning stage of 25 generations thus, during this period, each SD_k has been employed exactly 5 times. Then, at the end of the stage, each SD_k is updated according to:

$$SD_k = SD_k + U(1,5) \cdot (SD_b - SD_k) + U(-3,3) \tag{17}$$

where SD_b is the best SD_k (i.e. the one associated to the greater $succ_k$), $(\cdot)$ is the usual sign function returning -1, 0 or 1, and $U(a,b)$ denotes a uniform random integer in $[a,b]$.

In this way, each synchronization degree SD_k is updated partly in a pure random way and partly towards the best one of previous learning stage, i.e. SD_b. In the case that a SD_k takes a value outside the feasible domain, it is set at the violated bound (that is 1 or *NP*).

As in SAA1, the counter $succ_k$ is resetted after every update has been performed. Finally, note that the amounts of variations in the update rule (17) have been set after some preliminary experiments conducted with population size $NP = 100$.

6.3 SAA3: the deterministic strategy

In SAA3, conversely from the previous described schemes, a completely deterministic *SD* updating procedure is implemented.

This scheme has been introduced after an analysis of the impact of *SD* parameter on the population diversity shrinking rate. Indeed, the experiments conducted (see Section 7) show that small values of *SD* are able to speed up the diversity shrinking rate. Conversely, the population replacing procedure of the two employed base algorithms increases population diversity every 100 generations by replacing the worst 30% of the population (see Section 4.4). For these reasons we have decide to split the evolution into different stages, each one composed by 100 generations. Therefore, in the first generation of each stage SAA3 employs $SD = NP$ and linearly shades it until $SD = 1$ in the last generation of the stage. More formally $SD = \lceil NP - t \cdot \frac{NP-1}{100} \rceil$ where t is the current generation in the stage.

In this way, at the begin of each stage the replacement procedure increases population diversity allowing more search moves to DE and resetting SD to NP allows to maintain relatively higher this number of moves in the early generations of the stage while the SD shading toward 1 speed up the convergence by accelerating the diversity shrinkage until the next population partial replacement is performed. Finally note that this scheme implements the famous EAs rule-of-thumb that prefers exploration in early steps and exploitation in later steps [9].

7 Experimental Results

The performances of the previously proposed DE schemes have been evaluated on the first 15 benchmark functions proposed in [33] for the competition on real-parameters optimization at CEC 2005. This benchmark suite has been widely used for DE schemes comparison (see for example [2, 4, 25]) and it is composed by problems presenting a variegate mix of the properties of modality, separability, regularity, and differentiability.

Each benchmark is investigated with dimensionality $D = 10$. The allowed cap of fitness evaluations (NFES) is 100000, while an execution is regarded convergent if $f(x) - f(x^{opt}) \leq \varepsilon$, where ε is 10^{-6} for the unimodal functions $(f_1, \ldots, f_5)$ and 10^{-2} for the multimodal ones $(f_6, \ldots, f_{15})$.

Two different kind of experiments were conducted. First we have analyzed the behaviour of Asynchronous DE (ADE) without parameters adaptation, and then we have made a comparison among the new proposed self-adaptive schemes (SAA*) and their base algorithms (SaDE* and jDE-2) aiming at point out the performances improvements obtained with the introduction of a self regulatory population update scheme. In the following subsections a detailed description of these experiments is provided.

7.1 Analysis of Asynchronous DE

In the analysis of Asynchronous DE (ADE) conducted in [19], for each benchmark, different combinations of genetic parameters (including SD and SH) have been tested. The scale factor F and the crossover probability CR change from 0.2 to 1 at a step of 0.2. The population size NP lies within 20 and 100 at a step of 20. The DE learning strategies considered are "rand/1/bin" and "best/1/bin". The synchronization degree values tested are chosen relatively to population size and they are 1, $NP/4$, $NP/2$, $3 \cdot NP/4$, NP, $3 \cdot NP/2$ and $2 \cdot NP$. Finally, also the three ordering strategies, "static", "dynamic", and "best", have been employed.

For the convenience of the following description, we call the simulation of each set of genetic parameters a trial. For each trial 100 executions have been held in order to eliminate the randomness of the statistical results. In each trial, the success rate SR (i.e. the number of convergent executions above the total number of executions) and the average NFES of every convergent execution C are recorded. These two indices are also synthesized in the quality measure $Q_m = C/SR$ introduced in [12] and suggested in [33].

In Figures 1 and 2 the "quality graphs" for the more significant benchmarks are provided. In each graphs the Q_m measure values at the varying of the synchronization degree SD are reported. Furthermore, every graph contains six curves, one for each combination of "base vector choice"/"shuffle strategy". In this way it is possible to compare ADE with Dynamical Differential Evolution (DDES) [27] (reproduced by combination "best"/"static" when $SD = 1$) and with classical DE (reproduced by combinations */"static" and */"dynamic" when $SD = NP$). Finally note that the Q_m values reported in the graphs are those obtained from the best combination of the other genetic parameters, i.e. NP, F, and CR.

These graphs show that, although smaller values of SD generally improve the performances of the algorithm, there is no single synchronization degree clearly superior to other ones. The choice of the best vector as base vector used in DDES is always the best choice throughout the unimodal functions $(f_1, \ldots, f_5)$ while this is not true for the multimodal ones. In the comparison with classical DE the difference of performances is even more pronounced, since a DE equivalent combination, "rand"/"dynamic" with $SD = NP$, is the best choice only on f_7 and in the other cases it is often far from the optimum choice. Shuffle strategies "dynamic" and "best" are almost always better than the "static" strategy used in DDES and in classical DE. Finally,

note that super-synchronous *SD*s always led to performances degrade. Anyway, for further results discussions refer to [19].

In Figure 3, the graphs related to the population diversity evolution are reported for six selected benchmarks: three unimodal (f_1, f_2, f_3) and three multimodal (f_7, f_9, f_{11}). The population diversity is computed every offspring/trial generation as the upper quartile of the distribution of distances between all pairs of solutions [20]. To avoid side effects due to randomness, the provided results have been averaged over 25 executions.

The diversity graphs provided in Figure 3 clearly show that the diversity shrinking rate is inversely proportional to the choice of the *SD* value, both for unimodal that for multimodal problems. Super-synchronous settings, i.e. $SD > NP$, present a diversity curve too much higher and generally do not converge (i.e. do not reach an almost zero diversity) giving us an insight of why super-synchronous ADEs are so poorly performant. Instead, differences of the diversity shrinking rates among the settings of *SD* from 1 to *NP* empirically prove what it is asserted in Section 6, i.e. that small *SD*s accelerate diversity shrink so shifting the DE exploration/exploitation balance towards exploitation.

7.2 Comparison of Self-Adaptive Asynchronous DEs

Aiming at point out the performances improvements obtained with the introduction of self-adaptive synchronization parameters into classical (self) adaptive DE schemes, we have held comparisons among the base algorithms (SaDE*, jDE-2) and the proposed *SD* and *SH* adaptation schemes (SAA1, SAA2, SAA3) implemented on top of these two base algorithms. Moreover, results for sequential implementations of SaDE* and jDE-2 (Seq-SaDE* and Seq-jDE-2) have also been provided.

The population size has been set to $NP = 10 \cdot D = 100$ for all the algorithms, while other secondary settings have been discussed in Sections 5 and 6. In order to eliminate the randomness of the results, for each benchmark 25 executions have been held.

In Tables 1 and 2, for each experiment, the same performances measures used in the previous subsection have been reported, that is: success rate *SR*, convergence speed *C*, and quality measure Q_m. Instead, Tables 3 and 4 provide means (f_{avg}) and standard deviations (f_{std}) of the best fitness obtained in each experiment after the allowed cap of NFES and averaged over all the 25 executions.

Tables 1 and 2 clearly show that the proposed SAA* schemes present better results than their base algorithms SaDE* and jDE-2 throughout the entire suite of benchmarks. For unimodal problems ($f_1, \cdots, f_5$) the convergence speeds of SAA* schemes are comparable and are always better than that of SaDE* and jDE-2. However, it is for multimodal problems ($f_6, \cdots, f_{15}$) that the difference in performances is even more marked. Other than a general improvement in convergence speed, the SAA* schemes are able to reach a better success rate than that of their base algorithms (both in generational that in sequential version) in those functions that does not present a full *SR*, i.e. f_6, f_7, f_{11}, f_{12}, f_{15}. Since some of these functions are separable and other ones are non-separable, we can assert that the proposed SAA* schemes are very versatile, other than efficient, especially on those problems that present a certain landscape complexity. Finally, note that, in two cases (f_1 and f_9) for SaDE*-based schemes and one case for jDE-2-based schemes (f_1), the purely sequential versions converge faster than the others. This is likely due to the fact that f_1 and f_9 are notoriously good benchmarks for exploitative algorithms [33].

Also in Tables 3 and 4 we can see that SAA* schemes presents better results than SaDE*, Seq-SaDE*, jDE-2, and Seq-jDE-2. In benchmark f_3 only our proposed schemes reach exactly the optimal fitness value in every one of the 25 executions, while for the unsolved benchmarks (those ones with $SR = 0$ for every algorithm), i.e. f_8, f_{10}, f_{13}, f_{14}, it can be seen a slight positive improvement in favor of SAA* schemes. Finally, note that the SAA* standard deviations are always comparable with those of the respective base algorithms.

Furthermore, Table 8 reports, for each benchmark and for each performance measure, the best performant DE scheme tried in this experiments session. It clearly shows that: 1) jDE-2-based schemes are almost always more performant than SaDE*-based algorithms, 2) a SAA* scheme is not the better one only in 3 results on 48 (counted without considering the "none" cells), 3) SAA2-jDE-2,

i.e. the "moving values" strategy associated with jDE-2, seems to be the more performant and robust scheme over the tested benchmark suite.

Finally, for completeness, in Figure 4 the evolution graph of the average *SD* value (over 25 executions) has been reported for benchmark f_3 and f_9 (note that the first is unimodal while the second is multimodal) and for every SAA* scheme. While SAA3 follows the deterministic behaviour described in Section 6.3, an important comparison can be done between SAA1 and SAA2. Indeed SAA2, by continuously modifying the values of his candidate *SD*s, seems to converge towards a single synchronization degree after some iterations. Instead, this strong convergence does not appear in SAA1 that, respect to SAA2, at end of the evolution it will have a substantially greater diversity of the *SD*s employed.

8 Conclusion and Future Work

This paper introduces novel techniques for self regulating the synchronization degree of DE algorithm, which can be tuned from a completely asynchronous to a super-synchronous behaviour. The paper also describes the integration of the proposed techniques in the state-of-the-art self-adaptive DE schemes and their experimentation with accepted benchmarks. From the best of our knowledge this is the first work which proposed a fully self-adaptive ADE scheme.

ADE, introduced in [19], is a generalization of DE scheme for handling different synchronization mechanisms introduced by weakening the classical EA concept of generations synchronized by iterations. The *SD* parameter regulates how fast a new candidate individual will actually becomes part of the population, thus participating in the mutation/crossover/selection process. With this generalization the order of individuals in the population becomes relevant, thus ordering strategies have been also introduced and discussed.

New experimental results aiming at point out the dependence of the evolution of population diversity with respect to the synchronization degree have been provided. The first session of experiments shows that there is not a clearly superior choice of *SD* and *SH*. For this reason three adaptation strategies for *SD* and *SH* have been proposed and tested on a variegate suite of benchmarks.

After an analysis of the state-of-the-art of (self) adaptive non asynchronous DE proposals, we have decided to implement the proposed self-adaptive asynchronous (SAA) schemes on top of SaDE [26] and jDE-2 [3]. In this way we have obtained six different parameters-free DE schemes.

Experimental results on the proposed SAA* schemes clearly show that our proposal improves the performances of the two employed (self) adaptive DEs, thus promoting our new synchronization parameters *SD* and *SH* to be used in future adaptive DE models as well as in effective implementations.

Future lines of work will concern the investigation of the relationships between the synchronization degree and the population size, other than parallel implementation of the proposed self-adaptive asynchronous DE schemes, where ability of anticipating or postponing the synchronization points among generations appears to be particularly suitable for parallel and distribuited architectures.

References

[1] M.M. Ali and A. Törn. Population set-based global optimization algorithms: some modifications and numerical studies. *Computers & Operations Research*, 31(10):1703–1725, 2004.

[2] J. Arabas, Ł. Bartnik, and K. Opara. Dmea - an algorithm that combines differential mutation with the fitness proportionate selection. In *Differential Evolution (SDE), 2011 IEEE Symposium on*, pages 1–8. IEEE, 2011.

[3] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M.S. Maučec. Performance comparison of self-adaptive and adaptive differential evolution algorithms. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 11(7):617–629, 2007.

[4] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657, 2006.

[5] U.K. Chakraborty. *Advances in differential evolution*. Springer Verlag, 2008.

[6] S. Das, A. Konar, and U.K. Chakraborty. Two improved differential evolution schemes for faster

global search. *ACM-SIGEVO Proceedings of GECCO*, 5:991–998, 2005.

[7] S. Das and P.N. Suganthan. Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, (numb. 99):1–28, 2010.

[8] K.A. De Jong. An analysis of the behavior of a class of genetic adaptive systems. *Dissertation Abstracts International*, 36, 1975.

[9] K.A. De Jong. *Evolutionary Computation. A Unified Approac*. MIT Press, 2006.

[10] K.A. De Jong and J. Sarma. Generation gaps revisited. *Foundations of genetic algorithms*, 2:19–28, 1993.

[11] M.G. Epitropakis, V.P. Plagianakos, and M.N. Vrahatis. Balancing the exploration and exploitation capabilities of the differential evolution algorithm. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 2686–2693. IEEE, 2008.

[12] V. Feoktistov. *Differential evolution: in search of solutions*, volume 5. Springer-Verlag New York Inc, 2006.

[13] R.A. Fisher and F. Yates. Statistical tables for biological, agricultural and medical research. *Statistical tables for biological, agricultural and medical research.*, (Ed. 3.), 1949.

[14] S. Ghosh, S. Das, A.V. Vasilakos, and K. Suresh. On convergence of differential evolution over a class of continuous functions with unique global optimum. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, (99):1–18, 2011.

[15] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 9(6):448–462, 2005.

[16] U. Maulik and I. Saha. Modified differential evolution based fuzzy clustering for pixel classification in remote sensing imagery. *Pattern Recognition*, 42(9):2135–2149, 2009.

[17] E. Mezura-Montes, J. Velázquez-Reyes, and C.A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485–492. ACM, 2006.

[18] Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*, volume 19. Springer-verlag Berlin, 1992.

[19] A. Milani and V. Santucci. Asynchronous differential evolution. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE.

[20] J. Montgomery. The effects of different kinds of move in differential evolution searches. *Artificial Life: Borrowing from Biology*, pages 272–281, 2009.

[21] J. Montgomery and S. Chen. An analysis of the operation of differential evolution at high and low crossover rates. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–8. IEEE.

[22] F. Neri and V. Tirronen. Recent advances in differential evolution: a survey and experimental analysis. *Artificial Intelligence Review*, 33(1):61–106, 2010.

[23] W. Pedrycz. *Fuzzy control and fuzzy systems*. John Wiley & Sons, Inc., 1993.

[24] K.V. Price, R.M. Storn, and J.A. Lampinen. *Differential evolution: a practical approach to global optimization*. Springer Verlag, 2005.

[25] A.K. Qin, V.L. Huang, and P.N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 13(2):398–417, 2009.

[26] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1785–1791. IEEE, 2005.

[27] A. Qing. Dynamic differential evolution strategy and applications in electromagnetic inverse scattering problems. *Geoscience and Remote Sensing, IEEE Transactions on*, 44(1):116–125, 2006.

[28] A. Qing. *Differential evolution: fundamentals and applications in electrical engineering*. Wiley-IEEE Press, 2009.

[29] A. Qing and C.K. Lee. *Differential Evolution in Electromagnetics*, volume 4. Springer Verlag, 2010.

[30] B.Y. Qu and P.N. Suganthan. Novel multimodal problems and differential evolution with ensemble of restricted tournament selection. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE, 2010.

[31] J. Ronkkonen, S. Kukkonen, and K.V. Price. Real-parameter optimization with differential evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 506–513. IEEE, 2005.

[32] R. Storn and K. Price. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. *Journal of global optimization*, 11(4):341–359, 1997.

[33] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real–parameter optimization. *KanGAL Report*, 2005.

[34] K. Tagawa. A statistical study of the differential evolution based on continuous generation model. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 2614–2621. IEEE, 2009.

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

[36] J. Teo. Exploring dynamic self-adaptive populations in differential evolution. *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, 10(8):673–686, 2006.

[37] H. Wang, Z. Wu, S. Rahnamayan, and D. Jiang. Sequential de enhanced by neighborhood search for large scale global optimization. In *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pages 1–7. IEEE, 2010.

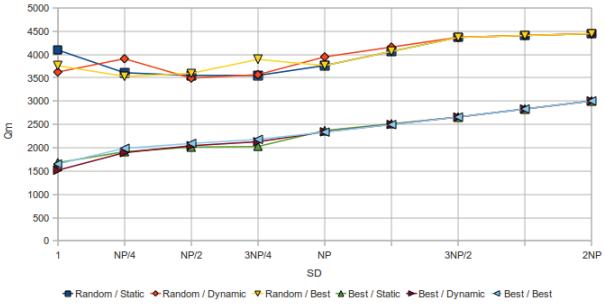
[38] D. Zaharie. Critical values for the control parameters of differential evolution algorithms. In *Proc. of MENDEL*, pages 62–67, 2002.

[39] D. Zaharie. Parameter adaption in differential evolution by controlling the population diversity. In *4th Int. Workshop Symbolic Numeric Algorithms Scientific Computing, Timi soara, Romania, October*, pages 9–12, 2002.

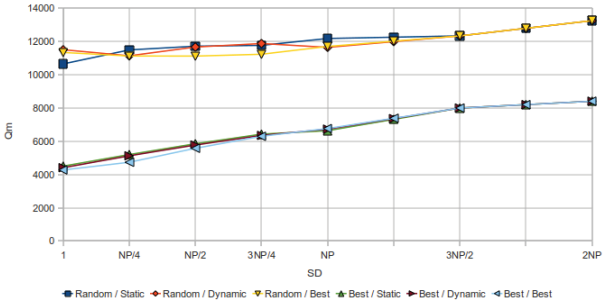
[40] D. Zaharie. Influence of crossover on the behavior of differential evolution algorithms. *Applied Soft Computing*, 9(3):1126–1138, 2009.

[41] J. Zhang and A.C. Sanderson. *Adaptive differential evolution: a robust approach to multimodal problem optimization*, volume 1. Springer Verlag, 2009.

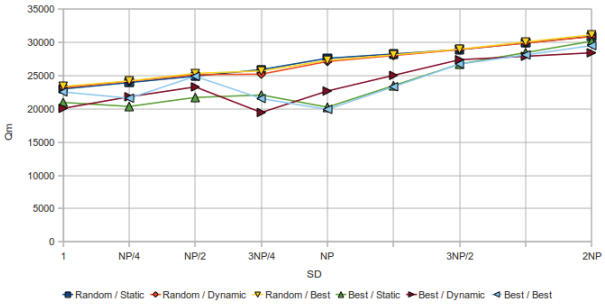
[42] J. Zhang and A.C. Sanderson. Jade: adaptive differential evolution with optional external archive. *Evolutionary Computation, IEEE Transactions on*, 13(5):945–958, 2009.



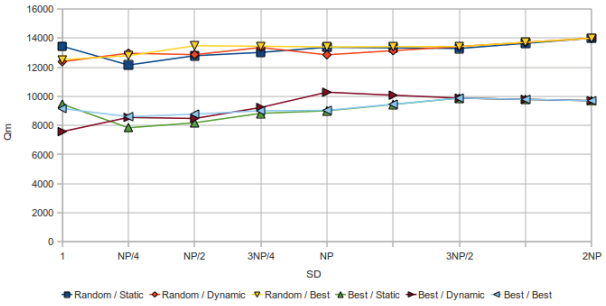
(a) f_1



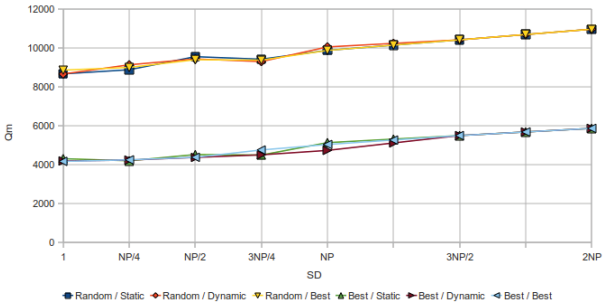
(b) f_2



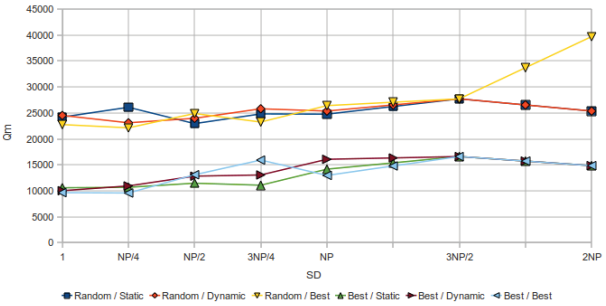
(c) f_3



(d) f_4

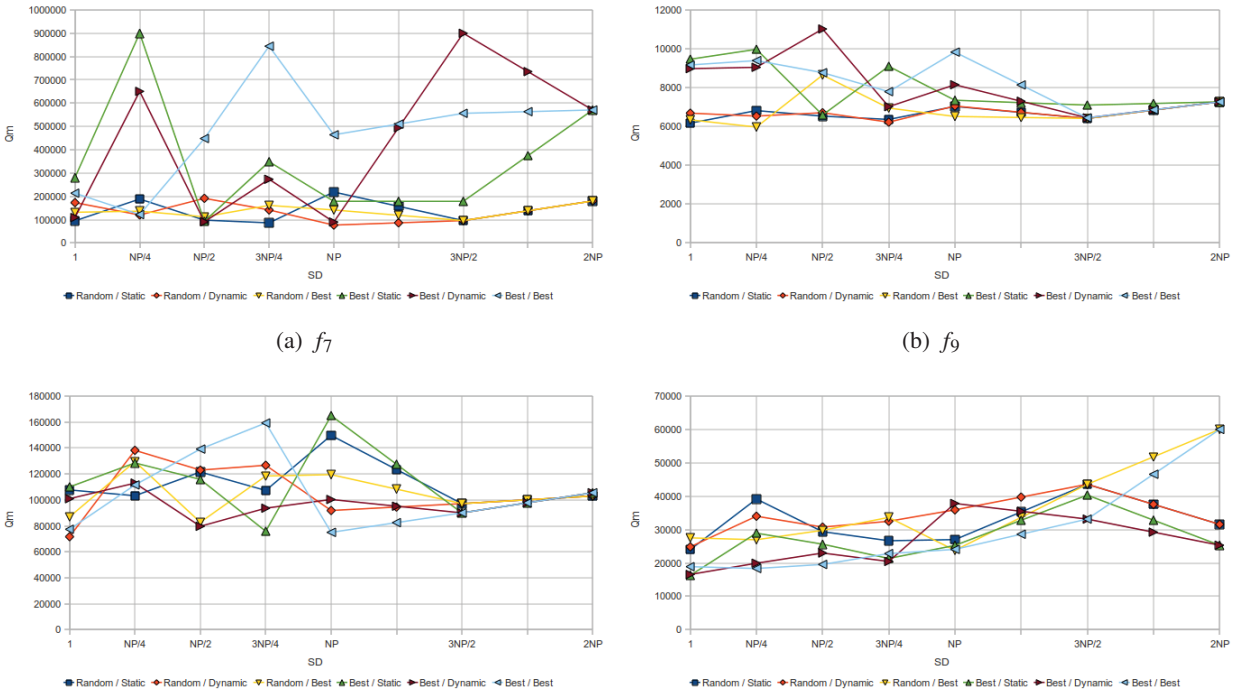


(e) f_5



(f) f_6

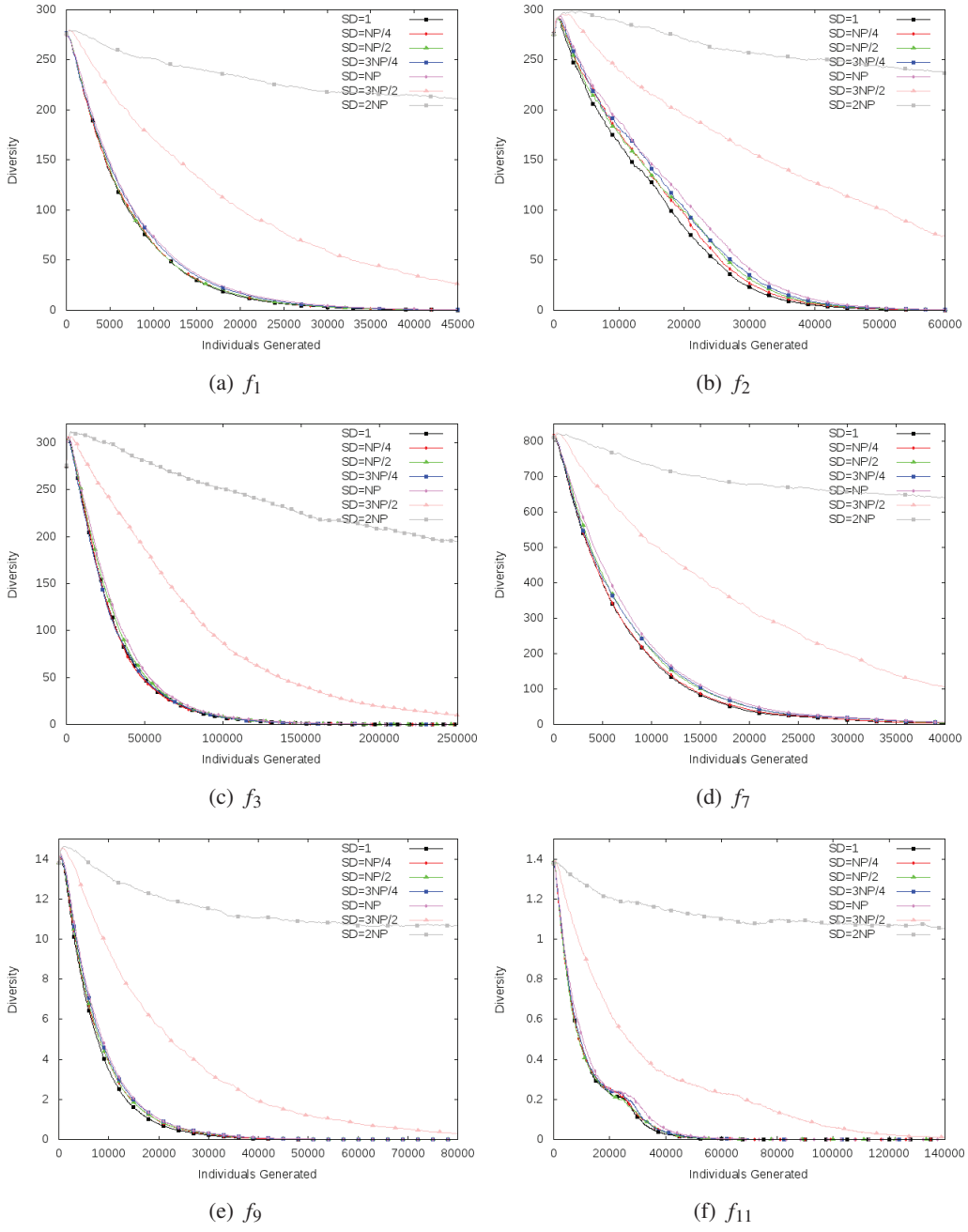
Figure 1. Q_m measure obtained varying NP and the couple ”base-vector” / ”shuffle-strategy”



(a) f_7 (b) f_9

(c) f_{11} (d) f_{12}

Figure 2. Q_m measure obtained varying NP and the couple ”base-vector” / ”shuffle-strategy”



(a) f_1 (b) f_2

(c) f_3 (d) f_7

(e) f_9 (f) f_{11}

Figure 3. Population diversity graph

Table 1. Success Rate and Convergence Speed for SaDE*-based algorithms

	SaDE*			**SAA1-SaDE***			**SAA2-SaDE***			**SAA3-SaDE***			**Seq-SaDE***		
f	SR	C	Q_m	SR	C	Q_m	SR	C	Q_m	SR	C	Q_m	SR	C	Q_m
f_1	1.00	24536	24536	1.00	23395	23395	1.00	23495	23495	1.00	23377	23377	1.00	22670	**22670**
f_2	1.00	56376	56376	1.00	54267	54267	1.00	52596	**52596**	1.00	52923	52923	1.00	54159	54159
f_3	1.00	54078	54078	1.00	53409	53409	1.00	52016	**52016**	1.00	52774	52774	1.00	53536	53536
f_4	1.00	56730	56730	1.00	56203	56203	1.00	55415	55415	1.00	55134	**55134**	1.00	55323	55323
f_5	1.00	62361	62361	1.00	59683	**59683**	1.00	61266	61266	1.00	60198	60198	1.00	59885	59885
f_6	0.84	80452	95776	0.84	77471	92227	0.88	80659	91657	0.92	79518	**86432**	0.80	77406	96757
f_7	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_8	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_9	1.00	55505	55505	1.00	56091	56091	1.00	55285	55285	1.00	55291	55291	1.00	54132	**54132**
f_{10}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{11}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{12}	0.32	64570	201781	0.20	66334	331669	0.40	67680	**169199**	0.24	69061	287754	0.32	65765	205515
f_{13}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{14}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{15}	0.40	75766	189414	0.40	77487	193717	0.48	79383	165381	0.56	81105	**144830**	0.36	77692	215811

Table 2. Success Rate and Convergence Speed for jDE-2-based algorithms

	jDE-2			**SAA1-jDE-2**			**SAA2-jDE-2**			**SAA3-jDE-2**			**Seq-jDE-2**		
f	SR	C	Q_m	SR	C	Q_m	SR	C	Q_m	SR	C	Q_m	SR	C	Q_m
f_1	1.00	17469	17469	1.00	15411	15411	1.00	15542	15542	1.00	16510	16510	1.00	15043	**15043**
f_2	1.00	26528	26528	1.00	25168	25168	1.00	24310	24310	1.00	24178	**24178**	1.00	24618	24618
f_3	1.00	49153	49153	1.00	44419	44419	1.00	41850	**41850**	1.00	43947	43947	1.00	47222	47222
f_4	1.00	28934	28934	1.00	27825	27825	1.00	26131	**26131**	1.00	27784	27784	1.00	28428	28428
f_5	1.00	50487	50487	1.00	45081	45081	1.00	44210	**44210**	1.00	47748	47748	1.00	45928	45928
f_6	0.84	78840	93857	1.00	76333	**76333**	0.96	75891	79053	0.88	75366	85643	0.92	70256	76365
f_7	0.00	–	–	0.04	67123	1678075	0.16	66598	**416237**	0.00	–	–	0.00	–	–
f_8	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_9	1.00	56511	56511	1.00	56568	56568	1.00	53472	**53472**	1.00	56686	56686	1.00	56360	56360
f_{10}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{11}	0.00	–	–	0.08	85398	1067475	0.16	79968	**499800**	0.04	68357	1708924	0.00	–	–
f_{12}	0.40	25430	63574	0.76	27990	36828	0.64	23123	**36129**	0.48	23887	49764	0.32	30486	95268
f_{13}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{14}	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–	0.00	–	–
f_{15}	0.44	69582	158140	0.44	67340	153045	0.64	59574	**93084**	0.36	68507	108741	0.32	63866	122819

Table 3. Fitness Mean and Standard Deviation for SaDE*-based algorithms

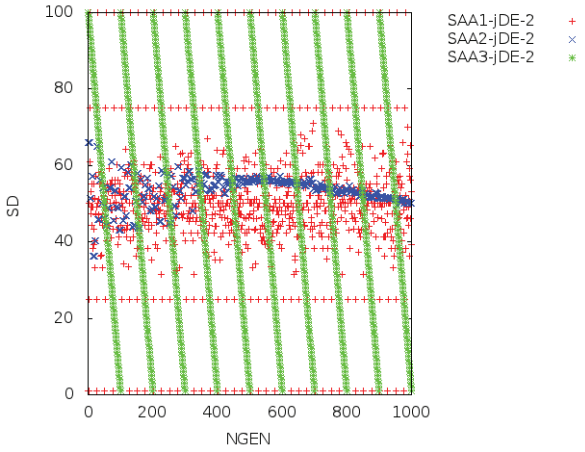
	SaDE*		**SAA1-SaDE***		**SAA2-SaDE***		**SAA3-SaDE***		**Seq-SaDE***	
f	f_{avg}	f_{std}	f_{avg}	f_{std}	f_{avg}	f_{std}	f_{avg}	f_{std}	f_{avg}	f_{std}
f_1	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
f_2	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
f_3	1.1102e−18	5.4389e−18	6.2172e−17	3.0458e−16	**0**	0	4.6629e−17	2.2843e−16	2.4424e−17	1.1965e−16
f_4	2.6645e−17	1.3053e−16	**0**	0	**0**	0	**0**	0	**0**	0
f_5	1.1144e−12	1.4723e−12	**5.2267e−13**	6.7305e−13	5.2067e−12	1.0175e−11	3.0193e−12	7.0921e−12	1.4483e−12	1.9065e−12
f_6	0.2103	0.5172	1.2269	3.7548	**0.0451**	0.1513	0.1806	0.6993	0.59687	1.4105
f_7	**0.0836**	0.0428	0.1039	0.0384	0.0916	0.0246	0.0963	0.0287	0.1050	0.0350
f_8	**20.3374**	0.0957	20.3517	0.0723	20.3433	0.0544	20.3631	0.0666	20.3551	0.0570
f_9	2.2766e−10	8.2403e−10	5.5431e−10	1.4401e−09	7.7041e−11	7.0494e−11	1.4757e−10	4.1397e−10	**3.3202e−11**	3.5723e−11
f_{10}	6.4609	1.6112	5.8444	1.0975	5.7051	1.5475	6.8287	1.8395	**5.7038**	1.2800
f_{11}	5.1632	0.6935	5.3098	0.6083	5.0311	0.8144	5.2764	0.6915	**5.0011**	0.8059
f_{12}	97.9075	304.4790	243.1590	474.5190	110.0370	326.4010	**58.0018**	140.5620	93.8410	264.8670
f_{13}	0.4142	0.0539	0.4112	0.0569	0.4258	0.0415	**0.4087**	0.0703	0.4118	0.0504
f_{14}	3.2019	0.1807	3.2456	0.1852	3.1488	0.1679	**3.1321**	0.1668	3.1908	0.19682
f_{15}	114.5900	167.1930	89.7238	146.0920	64.5462	146.4120	**32.3387**	108.4200	102.2050	156.9940

Table 4. Fitness Mean and Standard Deviation for jDE-2-based algorithms

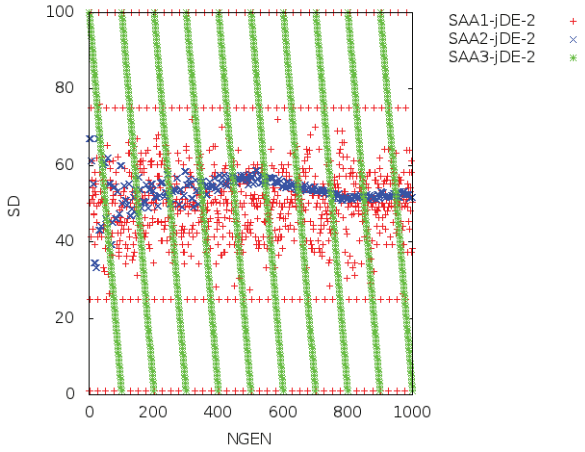
	jDE-2		**SAA1-jDE-2**		**SAA2-jDE-2**		**SAA3-jDE-2**		**Seq-jDE-2**	
f	f_{avg}	f_{std}	f_{avg}	f_{std}	f_{avg}	f_{std}	f_{avg}	f_{std}	f_{avg}	f_{std}
f_1	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
f_2	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
f_3	2.6662e−14	1.3061e−13	**0**	0	**0**	0	**0**	0	4.1718e−11	2.0437e−10
f_4	**0**	0	**0**	0	**0**	0	**0**	0	**0**	0
f_5	3.8369e−14	6.4915e−14	**7.1054e−16**	1.4862e−15	8.8817e−16	1.0986e−16	1.2008e−14	2.9106e−14	1.1297e−14	2.8878e−14
f_6	0.4793	1.2955	**0.0002**	0.0011	0.0006	0.0022	0.6347	1.3685	0.3189	1.0815
f_7	0.0662	0.0393	**0.0436**	0.0213	0.0460	0.0217	0.0843	0.0550	0.1096	0.0929
f_8	20.3650	0.0715	20.3388	0.0792	**20.3092**	0.0520	20.3490	0.0899	20.3449	0.0880
f_9	2.1316e−16	8.7259e−16	6.4392e−17	3.0485e−17	**0**	0	1.7763e−17	4.1451e−17	3.3306e−18	1.1957e−17
f_{10}	5.8597	2.3546	5.7462	2.3791	**5.6980**	1.6245	6.8219	2.2071	6.7305	2.0111
f_{11}	5.5066	2.5336	3.9630	1.0746	**2.9938**	1.1891	4.6060	2.1340	5.3637	1.4770
f_{12}	224.3810	528.9390	69.9111	331.5130	**69.7111**	352.4890	307.7210	604.8980	362.5440	602.5320
f_{13}	0.5656	0.1043	0.5910	0.1037	**0.5342**	0.0558	0.5785	0.0941	0.5401	0.0825
f_{14}	3.3170	0.2912	3.2002	0.1939	**3.0715**	0.2547	3.1456	0.2612	3.1396	0.2607
f_{15}	212.5410	197.1460	232.0000	193.3290	**162.4710**	98.8400	194.5410	161.5390	192.0000	199.8400

Table 5. Synthesis of the more performant schemes

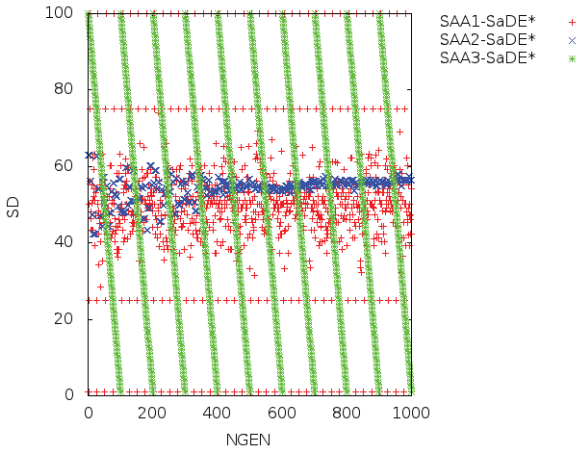
f	f_{avg}	SR	C	Q_m
f_1	all	all	Seq-jDE-2	Seq-jDE-2
f_2	all	all	SAA3-jDE-2	SAA3-jDE-2
f_3	SAA{1,2,3}-jDE-2, SAA2-SaDE*	all	SAA2-jDE-2	SAA2-jDE-2
f_4	all except SaDE*	all	SAA2-jDE-2	SAA2-jDE-2
f_5	SAA1-jDE-2	all	SAA2-jDE-2	SAA2-jDE-2
f_6	SAA1-jDE-2	SAA1-jDE-2	Seq-jDE-2	SAA1-jDE-2
f_7	SAA1-jDE-2	SAA2-jDE-2	SAA2-jDE-2	SAA2-jDE-2
f_8	SAA2-jDE-2	none	none	none
f_9	SAA2-jDE-2	all	SAA2-jDE-2	SAA2-jDE-2
f_{10}	SAA2-jDE-2	none	none	none
f_{11}	SAA2-jDE-2	SAA2-jDE-2	SAA3-jDE-2	SAA2-jDE-2
f_{12}	SAA3-SaDE*	SAA1-jDE-2	SAA2-jDE-2	SAA2-jDE-2
f_{13}	SAA3-SaDE*	none	none	none
f_{14}	SAA2-jDE-2	none	none	none
f_{15}	SAA3-SaDE*	SAA2-jDE-2	SAA2-jDE-2	SAA2-jDE-2



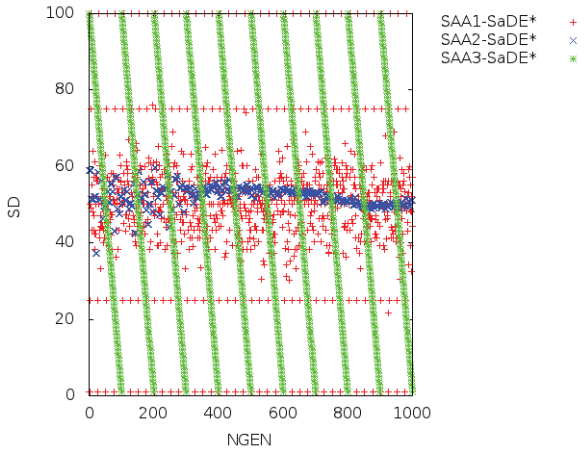
(a) jDE-2-based alg. on f_3



(b) jDE-2-based alg. on f_9



(c) SaDE*-based alg. on f_3



(d) SaDE*-based alg. on f_3

Figure 4. Evolution graph for parameter *SD*