# Parallel Evolutionary Algorithms Performing Pairwise Comparisons

Marie-Liesse Cauwet, Olivier Teytaud, Shih-Yuan Chiu, Kuo-Min Lin, Shi-Jim Yen, David L. Saint-Pierre, Fabien Teytaud

# Parallel Evolutionary Algorithms Performing Pairwise Comparisons

Marie-Liesse Cauwet, Olivier Teytaud
TAO, Inria, Lri, Umr Cnrs 8623
Bat. 650, Univ. Paris-Sud
91405 Orsay Cedex, France
cauwet@lri.fr

Shih-Yuan Chiu, Kuo-Min Lin, Shi-Jim Yen
Computer Science and Information Engineering
National Dong-Hwa University
Hualien, Taiwan

David L. St-Pierre
Industrial Engineering
Univ. du Québec at Trois-Rivières
Québec, Canada

Fabien Teytaud
Univ. Lille Nord de France, ULCO, LISIC, Calais, France
fabien.teytaud@lisic.univ-littoral.fr

## ABSTRACT

We study mathematically and experimentally the convergence rate of differential evolution and particle swarm optimization for simple unimodal functions. Due to parallelization concerns, the focus is on lower bounds on the runtime, i.e upper bounds on the speed-up, as a function of the population size. Two cases are particularly relevant: A population size of the same order of magnitude as the dimension and larger population sizes. We use the branching factor as a tool for proving bounds and get, as upper bounds, a linear speed-up for a population size similar to the dimension, and a logarithmic speed-up for larger population sizes. We then propose parametrizations for differential evolution and particle swarm optimization that reach these bounds.

## Categories and Subject Descriptors

G.1.6 [**Optimization**]: Unconstrained optimization

## General Terms

Theory

## Keywords

Differential Evolution, Parallelism, Particle Swarm Optimization

## 1. INTRODUCTION

Evolutionary algorithms are population-based algorithms designed to solve black-box optimization problems. An evolutionary algorithm typically (i) generates a population of search points (ii) selects a candidate solution for the optimization problem. The selection is performed thanks to the ranking/comparisons between the search points. The population makes parallelization highly relevant when the population is large: each point can be evaluated on a single processor [31, 16].

We first introduce parallel black-box optimization in Section 1.1. We then present the convergence rate of an algorithm in Section 1.2 before explaining what is the branching factor in Section 1.3. We then focus on parallel black-box optimization algorithms which use only "paired" comparisons, as explained in Section 1.4. Section 2 applies branching factor analysis to these algorithms, and provides lower bounds on runtimes. Sections 3 and 4 confront bounds with experimental rates for differential evolution and particle swarm optimization respectively.

Throughout the paper, unless stated otherwise, log refers to the natural logarithm.

### 1.1 Parallel black-box optimization

Typical optimization problems consist in searching for the minimum $x^*$ of some objective function $f : \mathbb{R}^D \to \mathbb{R}$, i.e. $x^*$ is such that $\forall x,\ f(x^*) \leq f(x)$. $f$ is also known as the fitness function and $D$ represents the dimension of the problem. A black-box optimization consists in doing so by successive calls to $f$ only. $f$ is seen as an oracle; no internal property of $f$ is used, and the goal of the optimization algorithm consists in finding a good approximation of $x^*$, within a moderate number of calls to the objective function.

Parallel synchronous black-box optimization is the setting in which $p$ simultaneous calls to the objective function are possible. Therefore, $p$ is the number of processors or cores, possibly on GPU architectures, which allow large numbers of cores. When measuring speed-up, i.e. the factor by which speed is improved when using $p$ processors, there are several cases: (i) Cases in which the objective function is fast. Then, the internal cost of the optimization algorithm and

possibly communication costs are not negligible. (ii) Cases in which the cost of the objective function is not approximately constant. Then, asynchronous effects matter. (iii) Cases in which the computational cost is mainly in the objective function, and the computational cost of the fitness function is constant. This leads to a synchronous case.

We focus on case (iii). Additionally, we consider that the structure of the optimization algorithm is not modified: we just use an increased population size, so that we can simultaneously evaluate one fitness value on each core or each processor. We will study the speed-up as a function of the population size, assuming that the population size is the number of cores. This is a special case, but a very usual one in practice.

## 1.2 Convergence Rate

We aim to study the effectiveness and limits of some evolutionary algorithms. As such, we define the *convergence rate* of an algorithm.
At each iteration $n$, the evolutionary algorithm provides an approximation $x_n$ of the optimum $x^*$. We consider the following convergence measure:

$$\lim_{n\to\infty} \frac{1}{n} \log \|x_n - x^*\| \to -C_{\lambda,D} \tag{1}$$

where $C_{\lambda,D}$ is a positive constant, depending on the population size $\lambda$ and the dimension $D$ only. Both theoretical and empirical results ([4, 5, 6, 27]) show that a wide range of evolutionary algorithms may converge in the sense of Eq. 1 when the objective function is unimodal and reasonably smooth. Our purpose is to exhibit some bounds on this convergence rate $C_{\lambda,D}$ when it is well defined. We also get lower bounds on the corresponding lim inf.

From the definition of the convergence rate (Eq. 1), it is straightforward that the speed-up of an evolutionary algorithm is equal to the convergence rate up to a multiplicative constant. Hence, in the following, results on the convergence rate also hold for the speed-up.

We have some extra informations available on the convergence rate $C_{\lambda,D}$, as follows [13].

(i) Convergence with a population size equal to the dimension, at best:

$$\textbf{Linear scalability: } \lim_{\lambda\to\infty} C_{\lambda,\lambda} = C_{linear}. \tag{2}$$

Eq. 2 is written for $C_{\lambda,\lambda}$. Indeed, Eq. 2 may also hold for $C_{\lambda,D(\lambda)}$ where $\lambda \to D(\lambda)$ is an approximately linear function, i.e. $D(\lambda)/\lambda = \Theta(1)$. [13, Section 5] has shown that general classes of comparison-based algorithms can not do better than such a convergence.

This equation means that we "cancel" the curse of dimensionality thanks to

- A parallel implementation with one individual evaluated on each processor.
- A linear number of processors (linear in the dimension).

(ii) Convergence with a large population size:

$$\textbf{Logarithmic speed-up: } \lim_{\lambda\to\infty} C_{\lambda,D} = C_{large,D} \log(\lambda) \tag{3}$$

$$\text{with } \lim_{D\to\infty} D \times C_{large,D} = C_{large} \tag{4}$$

For a wide class of comparison-based algorithms, including evolution strategies and pattern search methods, no algorithm can do better than this bound ([13, Fig. 2 and Prop. 8]).

We here show that these results also hold in the case of differential evolution and particle swarm optimization. Explicit bounds on $C_{linear}$ and $C_{large}$ are available thanks to the *branching factor* analysis sketched in Section 2.

The convergence rate above (Eq. 1) is a convergence rate in the search space; we can also consider the convergence rate in the fitness space:

$$\lim_{n\to\infty} \frac{1}{n} \log \Big( f(x_n) - \inf_x f(x) \Big) = -C'_{\lambda,D}. \tag{5}$$

On the sphere function, $C'_{\lambda,D} = 2C_{\lambda,D}$. We will use this convergence rate $C'_{\lambda,D}$ (in fitness space) in all our experiments.

## 1.3 Branching factor

Many evolutionary algorithms, after having evaluated the population, keep only a concise representation of the evaluations they have performed. This concise representation is typically:

- The ordered list of the indices of the $\mu$ best among the $\lambda$ generated individuals. There are $\binom{\lambda}{\mu} \times (\mu!)$ possibilities.
- The family (unordered) of the indices of the $\mu$ best among the $\lambda$ generated individuals. There are $\binom{\lambda}{\mu}$ possibilities.

A crucial property of such algorithms is that the number of possibilities is finite. The *branching factor* is this number. For evolution strategies, the branching factor can be strongly reduced, by assuming some regularity of the objective function (VC-dimension assumption on the objective function [35, 12]).

The branching factor is then used to prove that comparison-based algorithms have usually convergence rate increasing at most logarithmically with the population size $\lambda$ as in Eq. 3. This upper bound is proved for evolution strategies in [13, Fig. 2]. Moreover, this bound is tight: ad hoc variants of evolutionary algorithms match it. Thanks to some specific tuning, [32] shows empirically that this logarithmic speed-up can be reached by some evolutionary algorithms, namely CMSA, EMNA and CMAES. Similarly, the branching factor leads to finding a lower bound on the convergence rate provided that the population size is limited. Here again, this bound is tight, hence Eq. 2 holds for comparison-based algorithms (see [13, Section 5]).

## 1.4 Differential evolution and particle swarm optimization

Two families of evolutionary algorithms are studied in the present paper, namely Particle Swarm Optimization (PSO) and Differential Evolution (DE). They share several common points. They both are population based optimization algorithms and do not require heavy calculations for gradient or Hessian updates. Furthermore, the run of these algorithms depends on "paired" comparisons only: instead of comparing or sorting most of the population, these two algorithms compare each individual to only one possible mutation. This leads to specific mathematical properties, discussed in Section 2.

Among evolutionary algorithms, Differential Evolution was, from the very beginning [31], presented as a tool with high relevance for parallel optimization in continuous domains. Particle Swarm Optimization [16, 30] is also a general purpose black-box optimization algorithm, aimed at finding quickly approximate solutions to optimization problems.

In the implementation of these algorithms (see Algorithms 1 and 2), we use two loops per generation: one loop for computing fitness values of candidate vectors; one loop for updating "best" points (best search point so far, and best point so far for each given particle for PSO). One can implement variants of PSO and DE in which this is done in one loop only. However, such variants are less parallel, because the mutations, which depend on "best" points, might change during the loop and therefore cannot be computed simultaneously.

Choosing the right parameters for these algorithms is a challenging task. In this paper, we define rules for choosing these parameters in a parallel black-box optimization setting. We restrict this experimental part of the study to the case of unimodal well conditioned fitness functions. We validate our empirically developed formula for the parameters in front of both (i) standard values used for DE and PSO and (ii) theoretically known bounds.

**Differential evolution.** Differential evolution (DE) [31] is one of the main evolutionary algorithms for continuous optimization. It performs well on optimization testbeds [31, 3, 2, 24]. At each iteration, each point $p_i$ of the population undergo mutations of some of its components, then comparisons are performed between the old point and its corresponding mutated version. The best of the two points is selected for the next iteration. Different parameters are involved in the mutation step. A classical version of DE is presented in Algorithm 1. Many variants exist, including self-adaptive parameters [8, 17, 37, 25] and meta-heuristics for choosing parameters [23]. In some variants [31] (DE/best/1, DE/best/2), we need to compute at each generation the best search point so far. When this property is required, we set parameter *best* to 1 in Algorithm 1, and to 0 otherwise. Henceforward, when $best = 1$, we denote this version of DE by DE/best, and when $best = 0$, this version of DE is named DE/rand.

On the technical side, the large scale parallelization of the evaluations is performed with CUDA in [26]. [21] focuses on the asynchronous parallelization, which is not considered in the present paper; they combine DE and simulated annealing. [36] considers a ring topology and focuses on avoiding premature convergence. [38] considers the problem of multiple populations from the point of view of global convergence and maintaining diversity, while we, in the present paper, consider mainly (at least for experimental rates - our lower bounds on runtime are for arbitrary functions with unique global optimum) local convergence. Consistently with [1], we consider the performance as a function of the population size; we also prove lower bounds and discuss the optimal parametrization of DE with large population size.

**Algorithm 1** Pseudo-code of basic differential evolution. The first loop on the population contains the fitness evaluations, supposed to be the key part of the computational cost; it is fully parallel. For $j \in \{1, \dots, D\}$, $x_{i,j}$ denotes the $j^{th}$ coordinate of a vector $x_i \in \mathbb{R}^D$. $\mathcal{U}(a,b)^D$ stands for an uniform random variable in the domain $(a,b)^D$.

**Require:** $D$: Dimension, $N$: Number of generations, $F \in [0,2]$: Differential weight, $Cr \in [0,1]$: Crossover probability, $\lambda$: Population size, $best \in \{0,1\}$
  **for** each $i \in \{1, \dots, \lambda\}$ **do**
    Initialize $p_i \sim \mathcal{U}(-500, 500)^D$
    Compute $f(p_i)$
  **end for**
  **if** $best = 1$ **then**
    $g \leftarrow \arg\min_{p_1,\dots,p_\lambda} f(p_i)$
  **end if**
  $n = 1$
  **while** n<N **do**
    **for** $i \in \{1, \dots, \lambda\}$ parallel loop **do**
      Randomly draw $a$, $b$ and $c$ distinct in $\{1, \dots, i-1, i+1, \dots \lambda\}$
      Define $p'_i \leftarrow p_a + F(p_b - p_c)$
      Randomly draw $R \in \{1, \dots, D\}$
      **for** each $j \in \{1, \dots, D\}$ **do**
        $p''_{i,j} \leftarrow p'_{i,j}$ if $rand < Cr$ or if $j = R$
        $p''_{i,j} \leftarrow p_{i,j}$ otherwise
      **end for**
      Compute $f(p''_i)$
    **end for**
    **for** $i \in \{1, 2, \dots, \lambda\}$ **do**
      **if** $f(p''_i) < f(p_i)$ **then**
        $p_i \leftarrow p''_i$
        **if** $best = 1$ & $f(p_i) < f(g)$ **then**
          $g \leftarrow p_i$
        **end if**
      **end if**
    **end for**
    $n \leftarrow n + 1$
  **end while**
  **if** $best = 0$ **then**
    $g \leftarrow \arg\min_{p_1,\dots,p_\lambda} f(p_i)$
  **end if**
  **Return** $g$

**Particle Swarm Optimization.** Particle Swarm Optimization (PSO) is another classical evolutionary algorithm for continuous domain. Let $\mathcal{P}$ be a population of particles; for each $i \in \mathcal{P}$, $x_i \in \mathbb{R}^D$ is in the search space and the parti-

cle has a velocity $v_i \in \mathbb{R}^D$. Let $p_i$ be the best known position of the particle $i$ and $g$ be the best found solution so far. Algorithm 2 presents a basic PSO following this notation.

**Algorithm 2** Basic PSO. The first loop on the population contains the fitness evaluations, supposed to be the key part of the computational cost; it is fully parallel. Notations are the same as in Algorithm 1, ie. $x_j$ (resp. $x_{i,j}$) is the $j^{th}$ component of vector $x$ (resp. $x_i$).

**Require:** $D$: Dimension, $N$: Number of generations, $\mathcal{P}$: Population, $b_l$: Lower bound of the search space, $u_l$: Upper bound of the search space, $w$,$\phi_p$,$\phi_g$: Parameters
  **for** each particle $i \in \mathcal{P}$ **do**
    Initialize its position $x_i$ from a uniform distribution: $x_i \sim \mathcal{U}(b_l, b_u)^D$
    Initialize its best known position: $p_i \leftarrow x_i$
    Initialize its velocity: $v_i \sim \mathcal{U}(-|b_u - b_l|, |b_u - b_l|)^D$
  **end for**
  Initialize best solution: $g \leftarrow \arg\min_{x_i,\ i \in \mathcal{P}} f(x_i)$
  $n = 1$
  **while** $n < N$ **do**
    **for** each $i \in \mathcal{P}$  parallel loop **do**
      Pick $r_p$, $r_g \sim \mathcal{U}(0, 1)$
      **for** each $j \in \{1, \ldots, D\}$ **do**
        $v_{i,j} \leftarrow w v_{i,j} + \phi_p r_p (p_{i,j} - x_{i,j}) + \phi_g r_g (g_j - x_{i,j})$
      **end for**
      *Update* $x_i \leftarrow x_i + v_i$
      *Compute* $f(x_i)$
    **end for**
    **for** each $i \in \mathcal{P}$ **do**
      **if** $f(x_i) < f(p_i)$ **then**
        $p_i \leftarrow x_i$
        **if** $f(p_i) < f(g)$ **then**
          $g \leftarrow p_i$
        **end if**
      **end if**
    **end for**
    $n \leftarrow n + 1$
  **end while**
  **Return** $g$

It has several parameters leading to complicated dynamics; for example $w$ (see Algorithm 2) is often thought of as an inertia, but it can be negative or greater than 1. Therefore, many works are devoted to choosing optimally these parameters [22, 34, 11].

As explained before, PSO is relevant for parallelization. [14] uses speculative parallelization. Speculative parallelization is known to ensure an asymptotic logarithmic convergence rate (i.e. the optimal rate as described in Eq. 3), but it does not, in the general case, ensure optimality in large dimension (Eq. 2). They use the topology for defining a parallel variant of PSO, without necessarily increasing the population size. We will here consider the simple increasing of the population size. [20] focuses on the implementation part of a parallel PSO, in particular taking care of node failure and communication costs, thanks to MapReduce - we will here assume that communication costs can be neglected, e.g. due to expensive fitness functions. [18] uses a fuzzy controller for a parallel PSO, applied to a power dispatch problem; the fuzzy part generates rules for the simulations. Other experimental works on parallel-PSO include [29] (focusing on parallel global optimization, whereas we focus on unimodal functions) and [9] (detailing communication strategies, which are beyond the scope of the present paper, because we focus on the case in which most of the cost is in the fitness evaluations).

**Outline of this paper.** The branching factor methodology has never been applied to differential evolution and particle swarm optimization. This is the purpose of the theoretical part of this paper. In this paper:

- We show mathematically that, whatever may be the parametrization, differential evolution and particle swarm optimization can not do better than a logarithmic speed-up (Eq. 3) and a linear scalability (Eq. 2).
- We estimate empirically the convergence rate, in the case of the sphere function. These convergence rates, in the easy considered setting, match the lower bounds.
- We propose some variants designed to reach a good convergence rate when the population size is large.

## 2. MATHEMATICAL ANALYSIS

In [13], the *branching factor* was identified as a crucial component of a comparison-based algorithm. We present a general framework of optimization in Alg. 3.

**Algorithm 3** General framework of algorithms with bounded branching factor $K$.

Initialize some state $S$
**while** not finished **do**
  Generate (possibly randomly) a family $\mathcal{E}$ of individuals, using $S$ only.
  Let $\mathcal{I}$ be the extracted information, which depends on (i) the fitness function (ii) $S$ (iii) $\mathcal{E}$; we assume that $\mathcal{I}$ has values in a finite set $\{1, \ldots, K\}$.
  Update: $S \leftarrow update(S, \mathcal{I})$.
  Recommend: let $x = recom(S)$ be the approximation of the optimum proposed by the algorithm.
**end while**

Define, for the theorem below, $x_m$ the approximation of the optimum that it recommends after $m$ iterations in Algorithm 3. It is known, by the following theorem [33] that the runtime necessary for hitting the optimum with probability $1 - \delta$ and precision $\epsilon$ is lower bounded by a function of the branching factor.

THEOREM 1 (BRANCHING FACTOR THEOREM). *Assume that an optimization algorithm has branching factor $K$. Let $\mathcal{D}$ be the search domain. We define $\mathcal{F}$, $n_{\epsilon,\delta}$ and $N(\epsilon)$ as follow:*

- *We denote by $\mathcal{F}$ the set of objective functions on $\mathcal{D}$, i.e. functions from $\mathcal{D}$ to $\mathbb{R}$. Each function in $\mathcal{F}$ has a unique minimum $x^*(f)$ in $\mathcal{D}$. We assume that for each $x \in \mathcal{D}$, there is at least one objective function $f \in \mathcal{F}$ with optimum $x^*(f) = x$.*

- *Let $n_{\epsilon,\delta}$ be the minimum number of iterations in the While loop in Algorithm 3 such that, for any $f \in \mathcal{F}$, with probability $1-\delta$, $\|x_{n_{\epsilon,\delta}} - x^*(f)\| \leq \epsilon$.*
- *We define $N(\epsilon)$ the maximum integer $n$ such that there exist $n$ points $(x_1, \ldots, x_n) \in \mathcal{D}^n$ satisfying $\|x_i - x_j\| \geq 2\epsilon$ for any $i \neq j$.*

$$\textit{Then, } n_{\epsilon,\delta} \geq \frac{\log(1-\delta)}{\log K} + \frac{\log N(\epsilon)}{\log K}. \tag{6}$$

$N(\epsilon)$ is the so-called packing number. In [13], bounds on the branching factor are computed for evolution strategies based either on a full ranking of the population or on a selection of the $\mu$ best individuals. The bounds are improved when the objective function is simple, e.g. has bounded VC-dimension. The purpose of this section is to show a bound on the branching factor in the case of differential evolution (Algorithm 1) and particle swarm optimization (Algorithm 2). Lemma 1 exhibits such a bound.

LEMMA 1. *If the objective function is $f(x) = \|x - x^*\|^2$ over $\mathcal{D} = (0,1)^D$ for some $x^* \in \mathcal{D}$, or any composition $x \mapsto m(\|x - x^*\|^2)$ with $m$ increasing, then the branching factor of DE/rand (see Algorithm 1, best = 0) is at most:*

$$K \leq \min\{(\lambda+1)^D, 2^\lambda\}. \tag{7}$$

*With the same assumptions, the branching factor of DE/best (see Algorithm 1, best = 1) and PSO (see Algorithm 2) is at most:*

$$K \leq \min\{\lambda(\lambda+1)^D, \lambda 2^\lambda\}. \tag{8}$$

PROOF. First, let us give the proof when DE/rand is used. The right hand side of the min corresponds to the branching factor of $\lambda$ comparisons; there are $\lambda$ comparisons, hence $2^\lambda$ possible comparison results, hence the $2^\lambda$ bound. We just have to show the left hand side of the min. There are $\lambda$ comparisons at each iteration, each of them between two points. This does not follow the scheme in [13] because, there, the branching was directly a ranking of the $\mu$ best among $\lambda$ or a selection (without ranking) of the $\mu$ best among $\lambda$. Here, one of two points is selected, $\lambda$ times. When $p_1$ and $p_1''$ are compared, $p_1$ is selected if $\|p_1 - x^*\| < \|p_1'' - x^*\|$. Similarly, the result of the comparison of $p_i$ and $p_i''$ depends on the position of $x^*$ w.r.t the median hyperplane of $p_i$ and $p_i''$. Therefore, the final comparison result depends on in which cell, among the cells obtained by the arrangement of $\lambda$ hyperplanes, contains $x^*$. The branching factor is therefore upper-bounded by the number of cells obtained by $\lambda$ hyperplanes in $\mathbb{R}^D$. By Zaslavsky's Theorem (see e.g. [28, Th 4, p. 27]), this number of cells is upper bounded by $\sum_{i=0}^{D} \binom{\lambda}{i}$; this is upper bounded by $(\lambda+1)^D$, hence the expected result for DE/rand.

We now extend to DE/best and PSO, as follows:

- When DE/best is used, the best point is one of the $\lambda$ selected points; hence the additional $\lambda$ factor: beyond the $2^\lambda$ possible results of the pairwise comparisons, we have $\lambda$ possibilities for the best of the $\lambda$ selected points. Hence the expected result for DE/best.
- When PSO is used, the "global" best so far $g$ is one of the $\lambda$ "best so far per particle" $p_i$. Hence, there are $\lambda$ possibilities; and the expected result for PSO with best particle.

□

Please note that if $\phi_g = 0$, then the branching factor of PSO verifies the same bound as DE without best (e.g. DE/rand).

LEMMA 2. *For an arbitrary family $F$ of objective functions, the branching factor of DE/rand is at most*

$$K \leq 2^\lambda, \tag{9}$$

*and the branching factor of PSO and of DE/best is at most*

$$K \leq \lambda 2^\lambda.$$

PROOF. The $2^\lambda$ and $\lambda 2^\lambda$ parts of Lemma 1 do not use any property of $F$. □

PROPERTY 1 (RUNTIME OF DE/RAND). *If the objective function is $f(x) = \|x - x^*\|^2$ over $\mathcal{D} = (0,1)^D$ for some $x^* \in \mathcal{D}$, or any composition $x \mapsto m(\|x - x^*\|^2)$ with $m$ increasing, for DE/rand, the runtime $n_{\epsilon,\delta}$ is lower bounded:*

$$n_{\epsilon,\delta} \geq \frac{\log(1-\delta)}{D\log(\lambda+1)} + \frac{\log(\lceil 1/(2\epsilon)\rceil)}{\log(\lambda+1)}. \tag{10}$$

*Again when applying DE/rand, but with an arbitrary family $F$ of objective functions, if $\lambda = D$, the runtime $n_{\epsilon,\delta}$ is lower bounded:*

$$n_{\epsilon,\delta} \geq \frac{\log(1-\delta)}{\lambda\log(2)} + \frac{\log(\lceil 1/(2\epsilon)\rceil)}{\log 2}. \tag{11}$$

PROOF. Just plug the bound (Eq. 7, 8 and 9) on the branching factor above in Theorem 1, Eq. 6, and use $\log N(\epsilon) \geq D\log(\lceil 1/(2\epsilon)\rceil)$ when $\epsilon \to 0$ if the domain $\mathcal{D}$ is bounded with non-empty interior. □

PROPERTY 2 (RUNTIME OF PSO AND DE/BEST). *If the objective function is $f(x) = \|x - x^*\|^2$ over $\mathcal{D} = (0,1)^D$ for some $x^* \in \mathcal{D}$, or any composition $x \mapsto m(\|x - x^*\|^2)$ with $m$ increasing, for PSO and DE /best, the runtime $n_{\epsilon,\delta}$ is lower bounded:*

$$n_{\epsilon,\delta} \geq \frac{\log(1-\delta)}{\log(\lambda) + D\log(\lambda+1)} + \frac{D\log(\lceil 1/(2\epsilon)\rceil)}{\log(\lambda) + D\log(\lambda+1)}. \tag{12}$$

*Again when applying PSO or DE/best, but with an arbitrary family $F$ of objective functions, if $\lambda = D$, the runtime $n_{\epsilon,\delta}$ is lower bounded:*

$$n_{\epsilon,\delta} \geq \frac{\log(1-\delta)}{\log(\lambda) + \lambda\log(2)} + \frac{\log(\lceil 1/(2\epsilon)\rceil)}{1 + \log 2}. \tag{13}$$

PROOF. See proof of Property 1. □

**Remarks:** Eq. 10, 11, 12, 13 imply that if DE and PSO algorithms converge in the sense of Eq. 1 then

$$\lim_{\lambda \to +\infty} C_{\lambda,D} = O(\log(\lambda)). \quad (14)$$

$$\lim_{\lambda \to +\infty} C_{\lambda,\lambda} = O(1). \quad (15)$$

We have therefore proved that the differential evolution algorithm can not do better than $O(\log(\lambda))$ speed-up, when using $\lambda$ fitness evaluations per iteration. Eq. 11 and 13 do not prevent a linear speed-up until $\lambda = D$. In the case of evolution strategies, it is known that such a linear speed-up is possible; for DE and PSO, we are not aware of such a result. We will investigate this empirically in Sections 3 and 4.

# 3. EXPERIMENTS: SPEED-UP AND SCALABILITY OF PARALLEL DIFFERENTIAL EVOLUTION

We have shown mathematical lower bounds on the runtime when using differential evolution; hence, if Eq. 1 holds, this implies lower bounds on $CR = -C'_{\lambda,D}$ (see Eqs. 5, 14 and 15). We now check if these bounds are reached or approximately reached; we will see that this is approximately the case, as far as we can tell on (necessarily finite) experiments.

In all this section, experiments are performed on the sphere function $x \mapsto \|x\|^2$. We use 1000 iterations for evaluating the convergence rate. So now on, the convergence rate $CR$ is estimated as (see Eq. 5):

$$CR = \frac{1}{1000} \left( \log(\textit{best fitness at } 1000 \textit{ iterations}) - \log(\textit{initial best fitness}) \right).$$

Differential evolution depends on two parameters that take place in the mutation step of the algorithm: the differential weight $F$ and the crossover probability $Cr$. We are interested in finding the parameters $F$ and $Cr$ which enable to reach these lower bounds. We first do experiments with the baseline differential evolution DE/rand (Algorithm 1 with $best = 0$) in Section 3.1. We then perform additional experiments with large population sizes and extreme values of the parameters (Section 3.2). Finally, we test adaptive variants of differential evolution (Section 3.3).

## 3.1 Experiments with standard differential evolution

### 3.1.1 Testing $D = 5$ and large population size $\lambda$.

The purpose of this section is to experimentally check Eq. 3 by considering a fixed dimension $D = 5$. We increase the population size $\lambda$, and check if the convergence rate reaches the optimal $C'_{\lambda,D} = \Theta(\log(\lambda))$ rate. Section 2 has shown that it can not be better (see Eq. 14) than this $\log(\lambda)$ rate. Results presented in Figure 1 are averaged over 15 runs and the number of generations $N$ is fixed to 1000. The x-axis is the base-10 logarithm of the population size, and the y-axis shows the convergence rate $CR$, multiplied by $D$, as suggested in Eq. 4. Hence, if Eq. 3 holds, we should observe curves decreasing linearly with $log(\lambda)$.

We first experiment differential evolution with various constant parametrizations (i.e. parametrizations which are independent from the population size $\lambda$). Different parameters $F$ and $Cr$ are tested in Figure 1a. It appears that (i) the parameter $Cr$ seems to have no impact on the convergence rate $CR$; (ii) when the population size increases, a smaller parameter $F$ is better. For example, when $\log 10(\lambda) \in [1, 1.3]$, $F = 0.41$ is a good parameter choice. Then when $\log 10(\lambda) \in [1.3, 1.5]$, $F = 0.24$ becomes a better choice. Finally, when $\log 10(\lambda) \in [2.7, 4.4]$, $F = 0.07$ appears to be a better parametrization. That is why from now on we experiment with some differential evolution parametrizations with $F$ decreasing when the population size $\lambda$ increases. In Figure 1b, we investigate the case of parameter $F$ equal to $.5 \log(\lambda)^{-A}$ for various parameters $A$. Parameter $A$ fixed to 0.76 is seemingly a good choice, as the corresponding curve is linear for $\log 10(\lambda)$ large enough; whereas other choices of parametrizations lead to plateauing curves. Figure 1c displays results for various values of $F$ when $F = .5\lambda^{-A}$. For several values of $A$ ($A = 0.4$, $A = 0.11$), the curves have the expected behavior: quasi-linear, when $\log 10(\lambda)$ is large enough. The slopes seem also better than in the case of $F = .5 \log(\lambda)^{-0.76}$. Specifically, $F = .5\lambda^{-0.4}$ reaches the best slope of these experiments. Hence, a detailed look at results shows that, for population sizes as in these experiments,

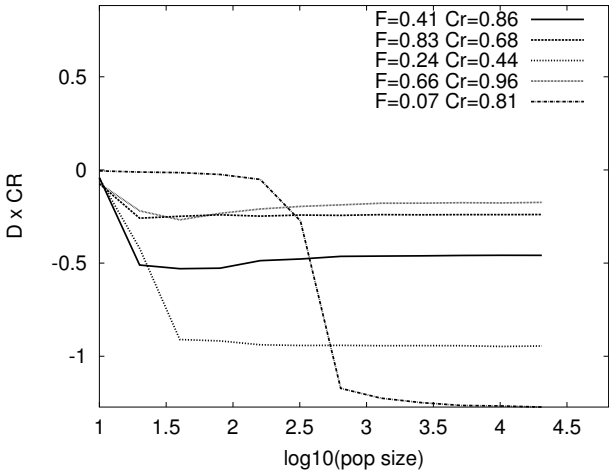
$$Cr = .44, F = .5\lambda^{-0.4} \quad (16)$$

is a reasonable tuning - though only mathematics could validate the (expected) logarithmic asymptotic convergence rate ($\simeq \log(\lambda)$) for $\lambda$ large.

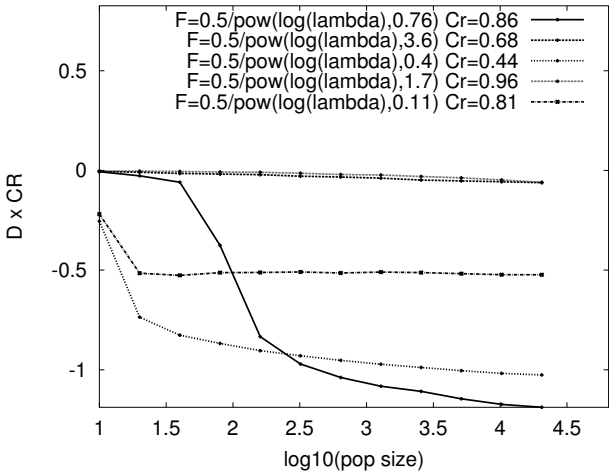
### 3.1.2 Testing $D = \lambda/2$, large population size $\lambda$.

The purpose of this section is to experimentally get Eq. 2. We set $D = \lambda/2$, and we observe what happens when $\lambda$ goes to infinity. Results are presented in Figure 2. Each experiment is launched 15 times and the number of generations $N$ is set to 1000. The x-axis is the base-10 logarithm of the population size, and the y-axis shows the convergence rate $CR$. We expect a negative constant convergence rate if Eq. 2 holds in the case of differential evolution (see Eq. 15).

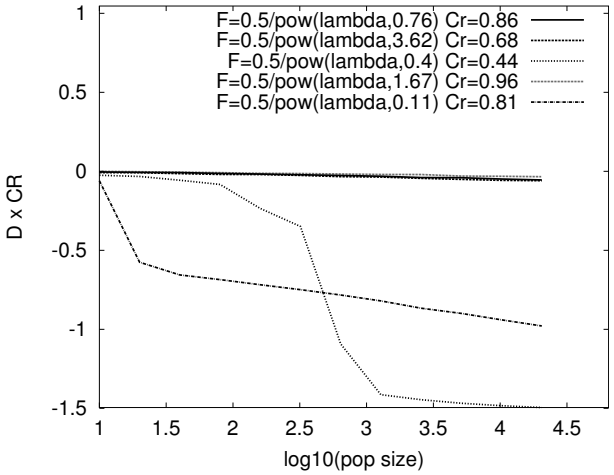
First, various constant parameters $F$ and $Cr$ are tested (see Figure 2a); in this case, none of the results are satisfactory as the curves converge toward 0: the algorithm does not converge. Figure 2b exhibit experiments when $F$ is inversely proportional to an exponent of $\log(\lambda)$. All but one converge toward 0 and the last one decreases linearly as a function of $\log(\lambda)$, which is also not the expected behavior. Last, parameter $F$ decreasing as an exponent of $\lambda$ is investigated (see Figure 2c). A seemingly good behavior happened when $F = .5\lambda^{-0.4}$; with this value, the curve approximately converges toward a constant value $-0.005$. With other tunings, either the curves converge toward 0 or decrease linearly. Crossover probability $Cr$ has seemingly no impact on the convergence rate $CR$. Experiments suggest that for $\lambda$ large and 1000 iterations the best performing tested variant is Eq. 16 ($Cr = .44$ and $F = .5\lambda^{-0.4}$), independently of the dimension.




(a) $F$ independent of $\lambda$. Seemingly, the convergence rate reaches a plateau.

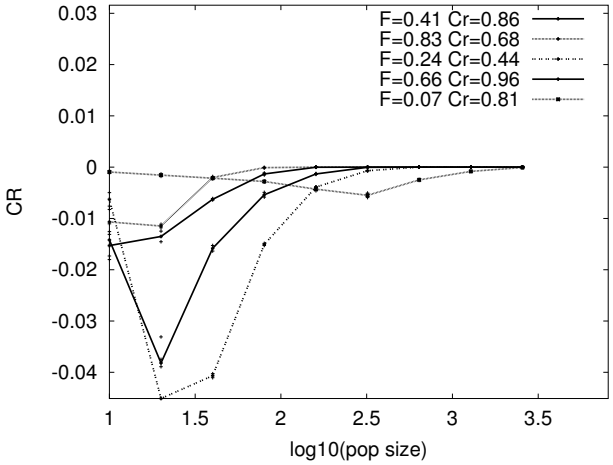



(b) $F$ inversely proportional to an exponent of $\log(\lambda)$. Results are better than above.

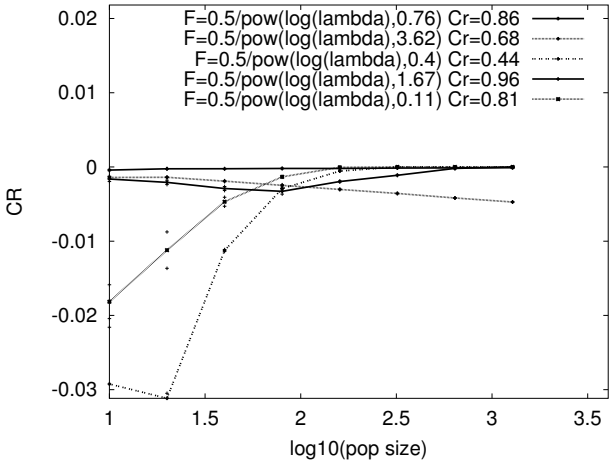



(c) $F$ inversely proportional to an exponent of $\lambda$. Results are better, in particular with $Cr = .44$ and $F = \frac{1}{2}\lambda^{-0.4}$.

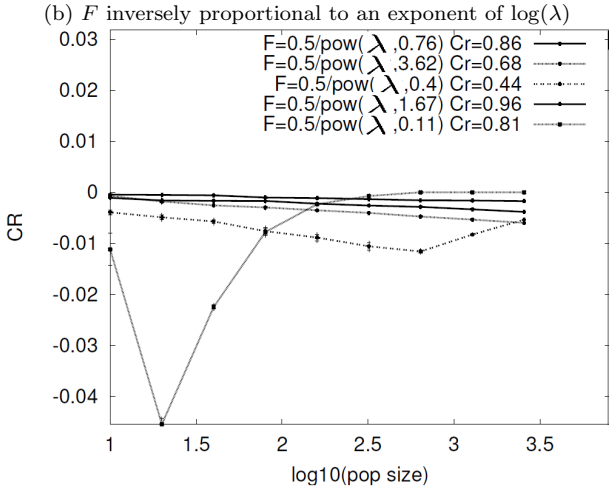
Figure 1: Experiments in dimension 5. Here we plot the convergence rates $CR$ (Eq. 5) of various parametrizations of differential evolution (estimated on 1000 iterations). All experiments are averaged over 15 runs. Standard deviations are very small and not presented. We essentially see that $F$ should decrease when $\lambda$ increases. Figure 3 will experiment with a few variants with larger population sizes.




(a) $F$ independent of $\lambda$




(b) $F$ inversely proportional to an exponent of $\log(\lambda)$




(c) $F$ inversely proportional to an exponent of $\lambda$

Figure 2: Here we plot the convergence rates (Eq. 5) of various parametrizations of differential evolution in the case $D = \lambda/2$ with $D$ the dimension. All experiments are averaged over 15 runs. Standard deviations are very small and not presented. Figure 3 presents experiments with a few selected variants with larger population sizes.

### 3.2 Extension: the case $F = 0$ and larger population sizes

Section 3.1 shows some surprisingly good results with $F$ very small. We investigate then the specific case $F = 0$. When we set $F = 0$, DE can still perform optimization, through the use of coordinate-wise mutations. This optimization does not converge asymptotically, because it is restricted to a finite set; yet, unless the number of iterations is huge, this is quite effective on the empirically measured convergence rate. This is shown in Figure 3, with a number of iterations $N$ set to 1000. Each result is the average of 15 runs. We present here experiments with larger population size, including $F = 0$.

Figure 3a displays the Convergence Rate in y-axis and the population size $\lambda$ in x-axis. Dimension $D$ is fixed to 3. Figure 3b presents experiments of Figure 3a with the Convergence Rate divided by $\log(\lambda)$ in y-axis and the population size $\lambda$ in x-axis. Hence, if Eq. 3 holds, we should observe a negative plateau if $\lambda$ is large enough. We notice that $F = 0$ is a bad parameter choice - even with a moderate number of iterations (1000)- as the convergence rate remains in 0 (the algorithm does not converge). Two parametrizations appear to perform well when the population size is large: $(F = .1, Cr = .5)$ and $(F = .5\lambda^{-0.4}, Cr = 0.44)$.

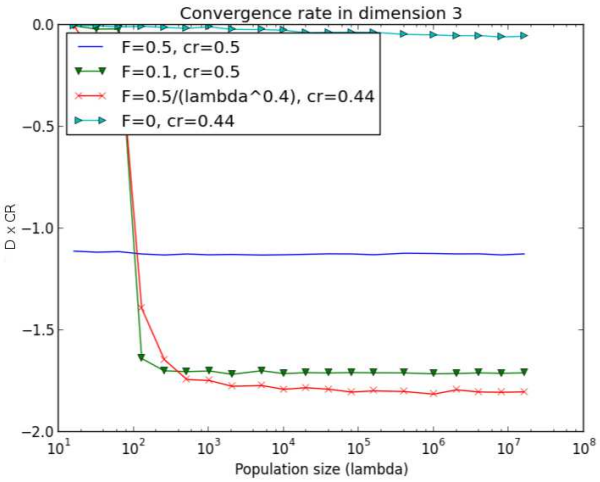
Figure 3c exhibits results when population size and dimension are linked. Dimension $D$ is here $\lambda/2$. We see that $F = 0$ is very satisfactory in such a case, and $F = .5\lambda^{-0.4}$ (which is also satisfactory in Figures 3a and 3b with fixed dimension) has the same performance in this $D = \lambda/2$ setting.

As a result, these experiments show that (i) $F = 0$ is suitable in the important case $\lambda$ proportional to the dimension and a fixed number of iterations. (ii) However, $F = 0$ is strongly suboptimal for a fixed dimension and $\lambda$ goes to infinity. (iii) $F$ decreasing such as $F = .5\lambda^{-0.4}$ independently of the dimension is seemingly a reasonably good idea in all cases. It is not clear whether it succeeds asymptotically for our two parallelization criteria (Eqs. 2 and 3 for $\lambda \to \infty$), but the behavior was better than other methods as far as we have seen on a simple sphere function problem.

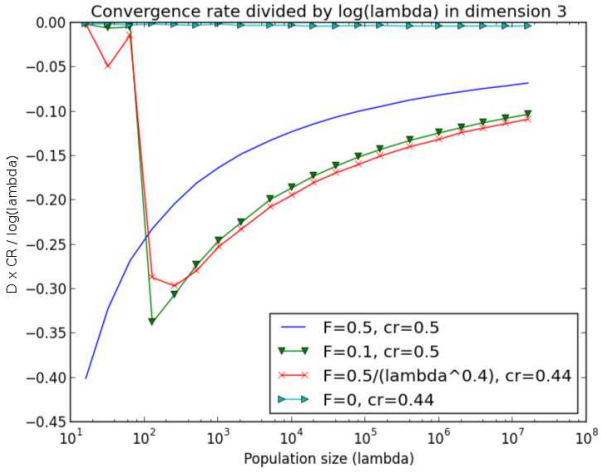
We see on these experiments that for a fixed total budget, when the dimension and/or the population size increase, the effect of the $F$ parameter is much more important than the $Cr$ value.

### 3.3 Adaptive DE

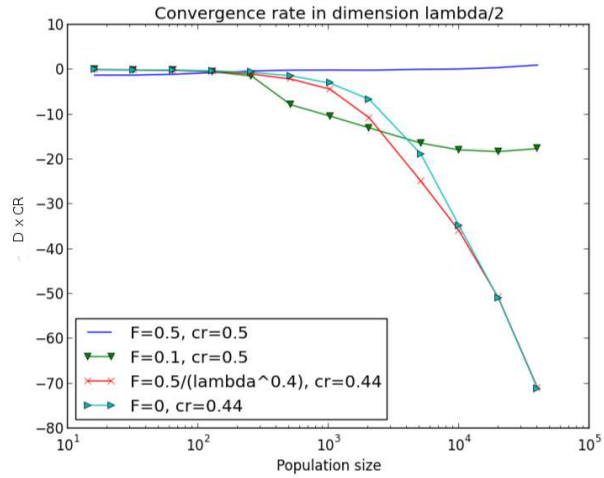
We here test adaptive variants of DE, aimed at choosing adaptively the $Cr$ and $F$ parameters. Experiments are performed with $F = 0.5$, $Cr = 0.9$, for DE/rand/1, DE/rand/2, DE/best/1, DE/best/2. The variants JADE and MDE_pBX are adaptive and choose their parameters themselves. We refer to [40, 15, 19] for more on the parametrization of differential evolution and the adaptive variants. Experimental results are presented in Figure 4 with 50 iterations. The adaptive variants of DE fail in preserving the $\log(\lambda)$ speed-up in case of large population size for a fixed $D$: the convergence rate decreases and is lower than for our non-adaptive variant (see Figure 4a). For $D = \lambda/2$, JADE succeeds reasonably well (see Figure 4b).




(a) Dimension $D = 3$, $D \times CR$ as a function of $\lambda$.

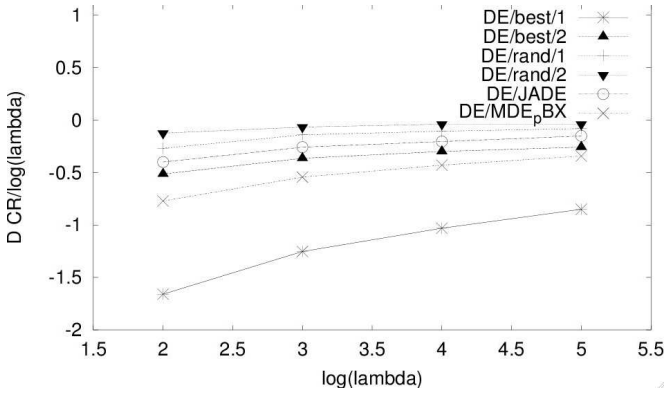



(b) Dimension $D = 3$, $D \times CR/\log(\lambda)$ as a function of $\lambda$.

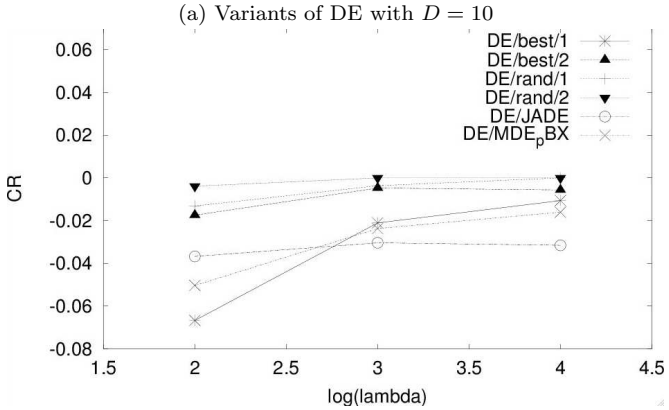



(c) Dimension $D = \lambda/2$, $D \times CR$ as a function of $\lambda$.

Figure 3: Convergence rate in the fitness space (Eq. 5) for large population sizes, including a parametrization with $F = 0$, some classical parametrizations, and our proposed formula $F = 0.5\lambda^{-0.4}$. Each point is averaged over 15 runs. These experiments performed in different settings are aimed at validating our proposed formula for large population sizes and checking that it is better than (i) some usual parametrizations (ii) the limit value $F = 0$. We see that $F = 0$ is not efficient for a fixed dimension: the algorithm does not converge. When the dimension is huge, then the case $F = 0$ performs well, whereas one can show mathematically that it does not converge. $F$ prescribed by $.5\lambda^{-0.4}$ performs well.




(a) Variants of DE with $D = 10$




(b) Variants of DE with $D = \lambda/2$

Figure 4: Experiments aimed at comparing our proposed variants with small values for $F$ as $\lambda \to \infty$ to adaptive variants of DE. In these experiments, variants of DE are tested with $D = 10$ and with $D = \lambda/2$ respectively, as the population size $\lambda$ goes to infinity. These results are convergence rates in the fitness space. Figure 4a shows that adaptive variants of DE can not compete with the standard DE (e.g. DE/best/1), which could not compete with our variants with $F = \frac{1}{2}\lambda^{-0.4}$ in previous experiments. Figure 4b shows that JADE performs reasonably well when the population size is proportional to the dimension.

# 4. EXPERIMENTS: SPEED-UP AND SCALABILITY OF PARALLEL PARTICLE SWARM OPTIMIZATION

We now try to reach the previous theoretical bounds in the case of PSO. The problem investigated in this section is the possibility to find parameters of PSO such that we get approximately these two good properties, namely linear scalability (Eq. 2) and logarithmic speed-up (Eq. 3). Since asymptotically everyone dies, we focus on the following non-asymptotic versions:

$$PR_D(\lambda) = \frac{D}{100}\left(\log(best\ fitness\ at\ 100\ iterations) - \log(initial\ best\ fitness)\right) \quad (17)$$

$$PR_{lin}(\lambda) = \frac{1}{100}\left(\log(best\ fitness\ at\ 100\ iterations) - \log(initial\ best\ fitness)\right) \text{ with } D = \lambda/2 \quad (18)$$

where $PR$ stands for Progress Rate. Implicitly, here, fitness values depend on $\lambda$, which is the population size.

We do experiments on simple unimodal functions, the sphere function $f(x) = \|x\|^2$, plus a validation on the cigar function.

The parameters of PSO are $w$, $\phi_g$ and $\phi_p$ (see Algorithm 2). We evaluate the impact of each parameter. For the sake of comparison, we need a standard PSO [39, 10, 7] as a baseline. We choose $\omega = 1/(2\log(2)) \simeq 0.72$, $\phi_p = \phi_g = 0.5 + \log(2) \simeq 1.2$.

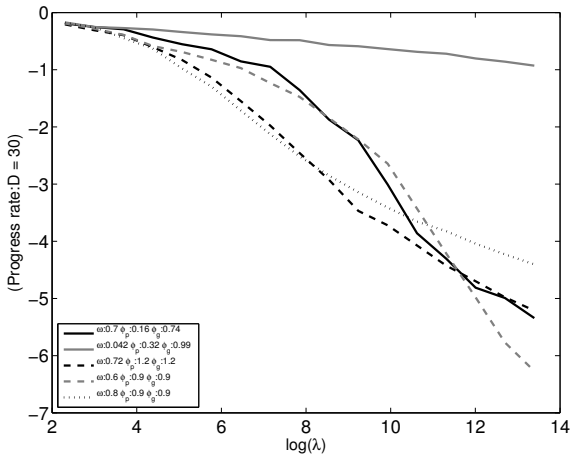
In Section 4.1, we investigate the case of large population size and logarithmic speed-up. Population size of the same order as the dimension and linear scalability are studied in Section 4.2.

## 4.1 Asymptotic scalability: large population size, fixed dimension

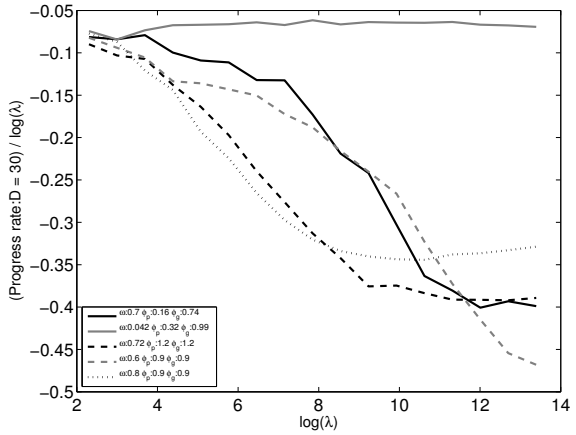
In this section we check if Eq. 3 holds. The fitness function used in these experiments shown in Figure 5 is $f(x) = \|x\|^2$. We limited the search space to an upper bound $b_u = 1$ and a lower bound $b_l = -1$. The number of generations $N$ is fixed to 100 and the dimension $D$ to 30. In Figure 5, the x-axis shows the natural log of the population size. Figure 5a displays $PR_{30}(\lambda)$ in order to show progress rate whereas Figure 5b displays $PR_{30}(\lambda)/\log(\lambda)$ to check if PSO reaches the optimal asymptotic behavior $PR_{30}(\lambda) = \Theta(\log(\lambda))$. If it is the case, the curves should show a plateau. Each experiment is repeated 10 times. Figure 5 seems to indicate that there exist different parameter values for which PSO reaches the optimal convergence rate (within a constant factor), at least as far as we can see in dimension 30. This comes a bit as a surprise as other algorithms struggle with such convergence rates (see [32] for efforts aimed at making some classical algorithms verify this optimal behavior) - standard PSO appears to be more naturally parallel than most existing evolutionary algorithms, from the point of view of the convergence rate as the population size grows to infinity, for simple unimodal functions. However, we will see below that for lower dimensions, things are not so nice for standard PSO.

**Choosing $\phi_g$ and $\phi_p$ in PSO:** Figures 6a, 6b and 6c look at the impact of $\phi_g$ and $\phi_p$ for different $\omega$ around these standard values. The x-axis still shows the population size expressed as $\log(\lambda)$ and the y-axis displays the progress rate. From Figure 6, it appears that $\phi_p$ and $\phi_g$ can only slightly change the convergence rate in our simple unimodal setting, which is in line with the current literature. From several empirical tests, in the following experiments we choose 0.9 instead of the recommended $0.5 + \log(2)$ as it seems to give better results regardless of the $\omega$ for the unimodal function under study in this paper.

**Choosing $w$ in PSO:** Figure 7 evaluates the impact of different $\omega$ for a given dimension $D = \{2, 3, 5, 10\}$, respectively Figures 7a, 7b, 7c ,7d, to see whether a dynamic $\omega$ could yield better results. As such, we are looking for curves that crosses one another. The x-axis still shows the population size expressed as $\log(\lambda)$ and the y-axis displays the progress rate. The first conclusion we can infer from Figure 7 is that the parameter $\omega$ clearly varies in relation to $\lambda$. Especially

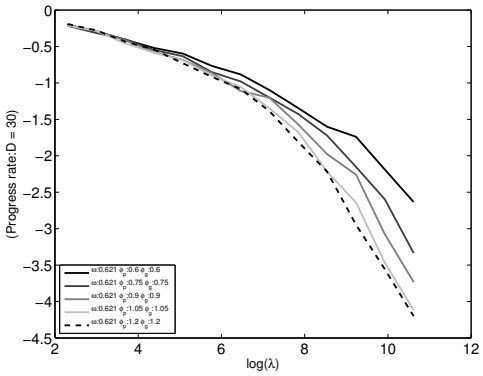



(a) Progress rate $PR_{30}(\lambda)$

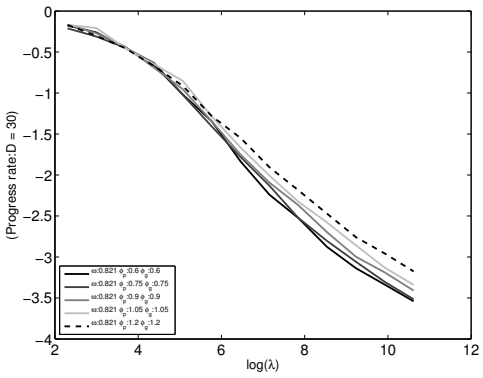



(b) Progress rate divided by $\log(\lambda)$, i.e. $PR_{30}(\lambda)/\log(\lambda)$

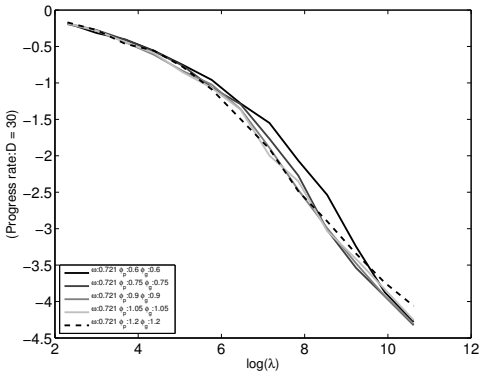
Figure 5: The progress rate, expressed as variants of Eq. (17), for different set of parameters $\omega$, $\phi_p$, $\phi_g$, including the values of standard PSO (black dash line with values $\omega = 0.72$ and $\phi_p = \phi_g = 1.2$). We observe that standard PSO performs well and even seems to reach the optimal convergence rate as $\lambda$ grows. For small $\lambda$ a higher value of $\omega$ is slightly more appropriate and eventually (around a population $\lambda = 80k$) smaller values of $\omega$ outperform standard PSO. Standard deviations are of order 0.01.




(a) $\omega = \frac{1}{2*\log(2)} - 0.1$




(b) $\omega = \frac{1}{2*\log(2)} + 0.1$




(c) $\omega = \frac{1}{2*\log(2)}$

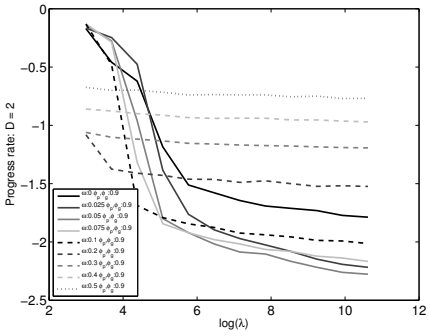
Figure 6: $PR_{30}(\lambda)$ for different values of $\omega$. In this figure, different values of $\phi_g$ and $\phi_p$ for values around those proposed in standard PSO are applied to the study of the (unimodal) sphere function $x \mapsto \|x\|^2$. We see that $\omega$ smaller (than standard PSO) is better for $\lambda$ large and $\omega$ larger is better for $\lambda$ small. The impact is, however, moderate in this case (dimension 30). Standard deviations are of order 0.01.

on lower dimensions (Figures 7a,7b, 7c and 7d), we observe patterns that emerge. For instance, in dimension $D = 5$, when the population size $\log(\lambda)$ is between $[2, 3]$, the best $\omega$ is 0.7. As the population size increases to $\log(\lambda) \in [3, 4]$, the best $\omega$ is equal to 0.6. At $\log(\lambda) \in [4, 5]$, the best $\omega$ becomes 0.5, etc. The rate at which the different curves crosses one another gives an important indication that there is a dependency of $\omega$ over both $\lambda$ and the dimension $D$.

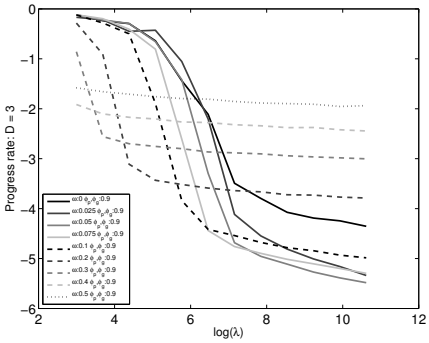
Figure 8 compares 3 different sets of parameters across dimensions $D = \{2, 3, 5, 50\}$, respectively Figures 8a, 8b, 8c, 8d. The first set of parameters are those provided by standard PSO. As we observe in Figure 7, as the population size $\lambda$ grows, smaller $\omega$ yields better performance. Thus, we can suppose that when $\lambda \to \infty$, $\omega = 0$ could be the best parameter. This is the second set of parameters with $\phi_p = \phi_g = 0.9$. The third one $\omega_\lambda$ is a set of parameters that depends on both $\lambda$ and the dimension $D$; this formula is extrapolated from the empirical results of Figure 7. It is given by:

$$\omega_\lambda = \max\left(0.025 + 4 * exp\left(-\frac{\lambda}{70}\right), 0.9 - \log\left(\frac{\lambda}{2.5 * D}\right)\right). \quad (19)$$

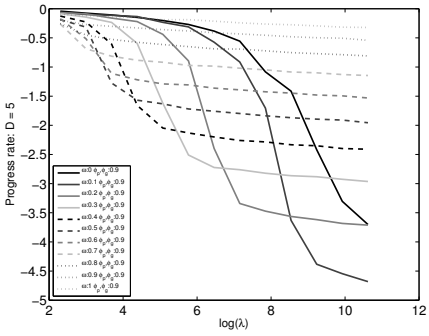
From Figure 8, it appears that using $\omega_\lambda$ instead of standard



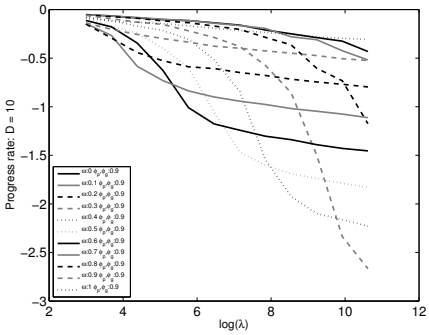

(a) Dimension 2, $\phi_g = \phi_p = 0.9$




(b) Dimension 3, $\phi_g = \phi_p = 0.9$




(c) Dimension 5, $\phi_g = \phi_p = 0.9$




(d) Dimension 10, $\phi_g = \phi_p = 0.9$

Figure 7: In dimension $D = \{2, 3, 5, 10\}$ we check the progress rate $PR_D(\lambda)$ for different values of $w$. In all cases, as $\lambda$ becomes large, smaller $\omega$ yield better results. The effect is more exacerbate on lower dimensions $\{2, 3, 5, 10\}$. The different values of $\omega$ produce curves that cross one another which indicate that there is a dependency over the population size $\lambda$ for the parameter $\omega$. We use these curves for designing rules for choosing $w$ as a function of the population size and the dimension (Eq. 19). Standard deviations are of order 0.01.

PSO yields results more robust in relation to different dimensions. In lower dimensions $D = \{2, 3, 5, 10\}$, the factor of improvement oscillates between 7 and 8. This means that using $\omega_\lambda$ instead of standard PSO in a parallel environment is up to 8 times more efficient. The parameters of standard PSO are essentially optimal for a dimension $D = 30$. It is important to note that $\omega_\lambda$ performs at least as good as standard PSO. As the dimension increases $D = 50$, $\omega_\lambda$ becomes again more efficient than standard PSO by a small margin. The special case $\omega = 0$ gives very good results in small dimensions $\{2, 3, 5\}$, improving over standard PSO by an average factor of 5, but never equals to $\omega_\lambda$ and rapidly becomes irrelevant in higher dimension $D > 10$ for the tested population sizes $\lambda$.

Figure 9 presents the same study as Figure 8 but using another unimodal fitness function, the Cigar. This fitness is defined as $f(x_i) = x_{i,0}^2 + 10^6 \sum_{d=1}^{D} x_{i,d}^2$. The use of $\omega_\lambda$ also appears to yield better results in relation to different dimensions. We observe the same behavior for $\omega = 0$ as in Figure 8.

## 4.2 Linear scalability: population size linear in the dimension

In this section we check if, for different variants of PSO, Eq. 2 holds. As a reminder, the fitness function used in these experiments is still $f(x) = \|x\|^2$. We limit the search space to an upper bound $b_u = 1$ and a lower bound $b_l = -1$. The number of generations $N$ is fixed to 100. The dimension $D$ is given by $\lambda/2$. Each experiment is repeated 10 times. Figure 10 presents the results for different parametrizations of $\omega$, $\phi_p$ and $\phi_g$.

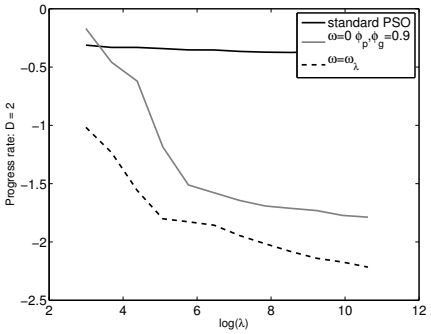
From Figure 10 it appears that again $\omega_\lambda$ yields the best progress rate. Standard PSO is following closely (in this setting; not in other settings as mentioned previously). The case where $\omega = 0$ is by far the worst setting of the 3 tested. Thus, we conclude that $\omega_\lambda$ seems to be a good parametrization for a parallel PSO environment, in terms of convergence rates.

# 5. CONCLUSION

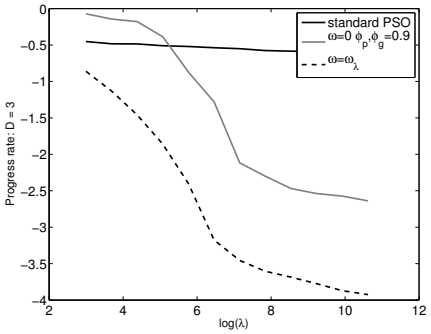
We have studied the speed-up of PSO and DE, as a function of the number of simultaneous function evaluations - a model of synchronous parallelism in which the main computational cost is in the objective function.

We have investigated (i) theoretically the lower bound on runtime for a large class of objective functions and (ii) empirically on the sphere function ( i.e. a model of unimodal well conditioned function) the convergence rate of differential evolution and particle swarm optimization when the population size $\lambda$ is large. Roughly speaking, the present paper extends [13] in the sense that it gives more precise bounds for DE and PSO, and it extends [32] in the sense that it provides (unproved) formulas for approximately optimal speed-ups for DE and PSO. The improvement compared to standard parametrizations grows seemingly indefinitely for DE, whereas it is rather limited for PSO where the standard parametrizations perform well even with huge values of $\lambda$.

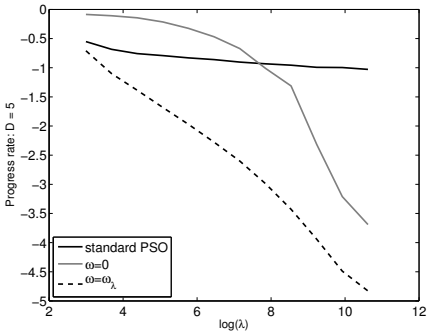
We have identified a similarity between PSO and DE: both



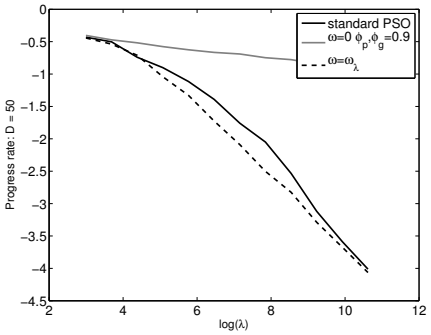

(a) Dimension 2




(b) Dimension 3

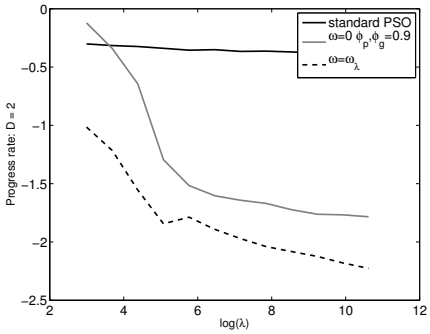



(c) Dimension 5

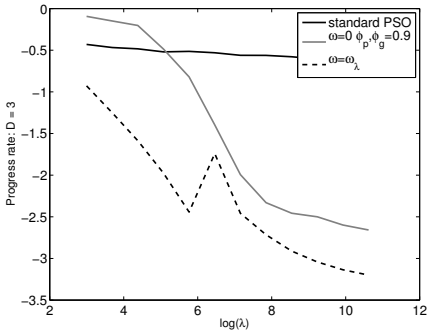



(d) Dimension 50

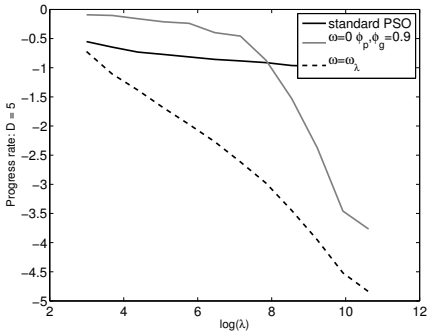
Figure 8: In dimension $D = \{2, 3, 5, 50\}$ respectively, we check the progress rate $PR_D(\lambda)$ for different values of $w$. At dimension $D = 2$, the use of $\omega_\lambda$ over standard PSO for parallelization yield an improvement by a factor of 7. At dimension $D = 3$, $\omega_\lambda$ improves over standard PSO by a factor of 8. At dimension $D = 5$, the factor is also 8. At dimension $D = 10$ (unpresented), the factor is 7.5. The case of dimension $D = 30$ (unpresented) provides no improvement, standard PSO is equal to $\omega_\lambda$. At dimension $D = 50$, there is a small advantage to use $\omega_\lambda$ over standard PSO. Standard deviations are of order 0.01.



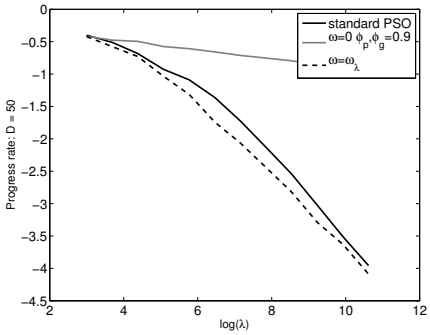

(a) Dimension 2




(b) Dimension 3




(c) Dimension 5




(d) Dimension 50

Figure 9: In dimension $D = \{2, 3, 5, 50\}$ respectively, we check the progress rate $PR_D(\lambda)$ for different values of $w$. At dimension $D = 2$, the use of $\omega_\lambda$ over standard PSO for parallelization yield an improvement by a factor of more than 5. At dimension $D = 3$, $\omega_\lambda$ improves over standard PSO by a similar factor. At dimension $D = 5$, the factor is around 9. At dimension $D = 10$ (unpresented), the factor is 2.5. The case of dimension $D = 30$ (unpresented) provides no improvement, standard PSO is essentially equal to $\omega_\lambda$. At dimension $D = 50$, there is a small advantage to use $\omega_\lambda$ over standard PSO. Standard deviations are of order 0.01.

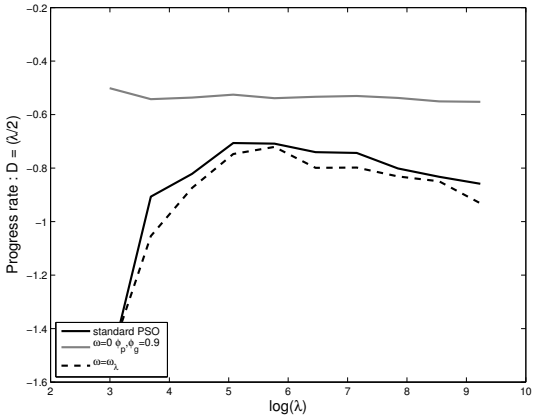



Figure 10: The performance (progress rate) for different population size $\lambda$ where the dimension is given by $\lambda/2$. Three sets of parameters are tested. The first set is the values given by standard PSO. The second test is given by $\omega = 0$, $\phi_g$,$\phi_p = 0.9$. The third and last set is the formula given by $\omega_\lambda$, $\phi_g$,$\phi_p = 0.9$. Again, we observe that $\omega_\lambda$ gives the best progress rate - though standard PSO is almost equivalent. Standard deviations are of order 0.01.

are based on pairwise comparisons. We have shown that DE and PSO:

- can not be faster than $-\frac{\log(\|x_n - x^*\|)}{n} = \Theta(\log(\lambda))$, with $x_n$ the best individual at iteration $n$, a fixed dimension, and $x^*$ the optimum;
- can not do better than $-\frac{\log(\|x_n - x^*\|)}{n} = \Theta(1)$ when the dimension is proportional to $\lambda$, e.g. $D \simeq \lambda/2$.

This mathematical lower bound on the runtime is for arbitrary families of fitness functions, provided that the optimum can be anywhere in the domain (i.e. for all $x$ in the domain, the family of functions contains at least one function with optimum $x$). The detailed bounds are improved when we consider the sphere function.

Interestingly, these rates are exactly the known limits for evolution strategies [13]. In the case of DE or PSO, we have no mathematical proof that these limits are reached. We checked experimentally whether some parametrization can be as fast as allowed by the mathematical bounds. Runtimes are estimated experimentally.

**Parametrization of differential evolution for large population sizes.** Experimentally, for differential evolution, a good parametrization might be $F \simeq .5\lambda^{-0.4}$ and some fixed $Cr$ (Eq. 16). Maybe such a parametrization reaches the bounds above. The improvement compared to standard parametrizations seemingly (on experiments) grows indefinitely as the population size goes to infinity.

**Parametrization of particle swarm optimization for large population sizes.** The $\omega$ parameter should decrease, whereas other parameters can be chosen close to those of standard PSO; for large population sizes, we get a factor 8 on the progress rate in dimension 5 thanks to the parametrization proposed in Eq. 19. Furthermore, standard PSO performs well on the important case $\lambda$ proportional to the dimension $D$, as well as when the dimension is 30; we had only minor improvements in these cases. Overall, PSO, under its standard forms, except for moderate dimensions (roughly from 2 to 10), performs well in a parallel setting with a large population size.

We certainly do not claim that Eq. 19 (for PSO) and Eq. 16 (for DE) are universal solutions for choosing parameters in PSO and DE respectively. They are relevant formulas for a clearly defined setting, namely a unimodal well conditioned setting with $\lambda$ large. Let us discuss limitations. First, this work is limited to unimodal well conditioned functions. Another limitation of this work is the choice of the number of iterations. We checked the convergence rates on limited numbers of iterations, which might be misleading. However, we believe that in a practical world, our population sizes (corresponding to parallelization on hundreds of thousands of cores - quite a big number even for clusters of machines equipped with GPU) and iteration numbers make sense.

**Further work.** We have shown runtime lower bounds for a class of algorithms including differential evolution and particle swarm optimization, for large population sizes. We have shown empirically that the runtime is close to that bound, for simple problems. This suggests that the runtime lower bounds are tight. Proving a corresponding upper bound on the runtime, matching the lower bound, is the main further work.
Designing adaptive rules which provide good rates for a wide class of fitness functions, beyond the simple unimodal scenario in the present paper, is also part of the agenda. Parametrization of $\omega$ (resp. $F$) proposed for PSO (resp. DE) might be improved, using some information on the budget, or maybe parameters depending on the iteration index; mathematical analysis of these formulas and/or improved variants is an interesting challenge.
Another natural extension is to adapt the bound for differential evolution variants with topology (e.g. [36]).

# 6. REFERENCES


[1] J. Arabas, O. Maitre, and P. Collet. Parade: A massively parallel differential evolution template for easea. In L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, and J. Zurada, editors, *Swarm and Evolutionary Computation*, volume 7269 of *Lecture Notes in Computer Science*, pages 12–20. Springer Berlin Heidelberg, 2012.

[2] D. Ardia, J. O. Arango, and N. G. Gomez. Jump-diffusion calibration using Differential Evolution. *Wilmott Magazine*, 55:76–79, 2011.

[3] D. Ardia, K. Boudt, P. Carl, K. M. Mullen, and B. G. Peterson. Differential Evolution with DEoptim: An application to non-convex portfolio optimization. *The R Journal*, 3(1):27–34, 2011.

[4] A. Auger. Convergence results for (1,λ)-SA-ES using the theory of $\varphi$-irreducible Markov chains. *Theoretical Computer Science*, 334(1-3):35–69, 2005.

[5] A. Auger, M. Jebalia, and O. Teytaud. (x,sigma,eta) : quasi-random mutations for evolution strategies. In *EA*, page 12p., 2005.

[6] H.-G. Beyer. *The Theory of Evolution Strategies*. Natural Computing Series. Springer, Heidelberg, 2001.

[7] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *IEEE Swarm Intelligence Symposium*, pages 120–127, 2007.

[8] 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. *Trans. Evol. Comp*, 10(6):646–657, Dec. 2006.

[9] J.-F. Chang, S.-C. Chu, J. F. Roddick, and J.-S. Pan. A parallel particle swarm optimization algorithm with communication strategies. *J. Inf. Sci. Eng.*, 21(4):809–818, 2005.

[10] M. Clerc. Beyond standard particle swarm optimisation. *IJSIR*, 1(4):46–61, 2010.

[11] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002.

[12] L. Devroye, L. Györfi, and G. Lugosi. *A probabilistic Theory of Pattern Recognition*. Springer, 1997.

[13] H. Fournier and O. Teytaud. Lower Bounds for Comparison Based Evolution Strategies using VC-dimension and Sign Patterns. *Algorithmica*, 2010.

[14] M. Gardner, A. W. McNabb, and K. D. Seppi. A speculative approach to parallelization in particle swarm optimization. *Swarm Intelligence*, 6(2):77–116, 2012.

[15] S. M. Islam, S. Das, S. Ghosh, S. Roy, and P. N. Suganthan. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(2):482–500, 2012.

[16] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, 1995.

[17] J. Liu and J. Lampinen. A fuzzy adaptive differential evolution algorithm. *Soft Comput.*, 9(6):448–462, June 2005.

[18] B. Mahdad, K. Srairi, T. Bouktir, and M. Benbouzid. Fuzzy Controlled Parallel PSO to Solving Large Practical Economic Dispatch. In IEEE, editor, *Proceedings of the 2010 IEEE International Conference of the IEEE Industrial Electronics Society*, pages 2695–2701, Phoenix, United States, Nov. 2010. IEEE.

[19] R. Mallipeddi, P. Suganthan, Q. Pan, and M. Tasgetiren. Differential evolution algorithm with ensemble of parameters and mutation strategies. *Applied Soft Computing*, 11(2):1679 – 1696, 2011. <ce:title>The Impact of Soft Computing for the Progress of Artificial Intelligence</ce:title>.

[20] A. McNabb, C. Monson, and K. Seppi. Parallel pso using mapreduce. In *Evolutionary Computation, 2007. CEC 2007. IEEE Congress on*, pages 7–14, 2007.

[21] J. Olensek, T. Tuma, J. Puhan, and Á. Bürmen. A new asynchronous parallel global optimization method based on simulated annealing and differential evolution. *Appl. Soft Comput.*, 11(1):1481–1489, 2011.

[22] K. E. Parsopoulos and M. N. Vrahatis. Parameter selection and adaptation in unified particle swarm optimization. *Mathematical and Computer Modelling*, 46(1-2):198–213, 2007.

[23] M. E. H. Pedersen. Tuning & simplifying heuristical optimization. January 2010.

[24] P. Pošík and V. Klemš. Jade, an adaptive differential evolution algorithm, benchmarked on the bbob noiseless testbed. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, GECCO Companion '12, pages 197–204, New York, NY, USA, 2012. ACM.

[25] K. V. Price, R. M. Storn, and J. A. Lampinen. *Differential Evolution - A Practical Approach to Global Optimization*. Natural Computing. Springer-Verlag, January 2006. ISBN 540209506.

[26] A. K. Qin, F. Raimondo, F. Forbes, and Y. S. Ong. An improved cuda-based implementation of differential evolution on gpu. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 991–998, New York, NY, USA, 2012. ACM.

[27] I. Rechenberg. *Evolutionstrategie: Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution*. Fromman-Holzboog Verlag, Stuttgart, 1973.

[28] E. Samansky. Zaslavsky's theorem. *Univ. Rice website*, 2002.

[29] J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka, and A. D. George. Parallel global optimization with the particle swarm algorithm. *JOURNAL OF NUMERICAL METHODS IN ENGINEERING*, 61:2296–2315, 2003.

[30] Y. Shi and R. C. Eberhart. A Modified Particle Swarm Optimizer. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 69–73, Washington, DC, USA, May 1998. IEEE Computer Society.

[31] R. Storn and K. Price. Differential evolution: A simple and efficient heuristic for global optimization over continuous spaces. *J. of Global Optimization*, 11(4):341–359, Dec. 1997.

[32] F. Teytaud and O. Teytaud. Log(lambda) Modifications for Optimal Parallelism. In *Parallel Problem Solving From Nature*, Krakow, Pologne, Sept. 2010.

[33] O. Teytaud and S. Gelly. General lower bounds for evolutionary algorithms. In $10^{th}$ *International Conference on Parallel Problem Solving from Nature (PPSN 2006)*, 2006.

[34] I. C. Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317 – 325, 2003.

[35] V. N. Vapnik. *The Nature of Statistical Learning*. Springer Verlag, 1995.

[36] M. Weber, F. Neri, and V. Tirronen. Parallel random injection differential evolution. In C. Di Chio, S. Cagnoni, C. Cotta, M. Ebner, A. EkÃąrt, A. Esparcia-Alcazar, C.-K. Goh, J. Merelo, F. Neri, M. PreuÃY, J. Togelius, and G. Yannakakis, editors,

*Applications of Evolutionary Computation*, volume 6024 of *Lecture Notes in Computer Science*, pages 471–480. Springer Berlin Heidelberg, 2010.

[37] M. Yang, J. Guan, Z. Cai, and L. Wang. Self-adapting differential evolution algorithm with chaos random for global numerical optimization. In Z. Cai, C. Hu, Z. Kang, and Y. Liu, editors, *Advances in Computation and Intelligence*, volume 6382 of *Lecture Notes in Computer Science*, pages 112–122. Springer Berlin Heidelberg, 2010.

[38] W.-j. Yu and J. Zhang. Multi-population differential evolution with adaptive parameter control for global optimization. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 1093–1098, New York, NY, USA, 2011. ACM.

[39] M. Zambrano-Bigiarini, M. Clerc, and R. Rojas. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In *IEEE Congress on Evolutionary Computation*, pages 2337–2344. IEEE, 2013.

[40] J. Zhang and A. C. Sanderson. Jade: adaptive differential evolution with optional external archive. *Trans. Evol. Comp*, 13(5):945–958, Oct. 2009.