Universiteit Leiden
The Netherlands

Opleiding Informatica

Performance Comparison of Configurable Particle Swarm and Differential Evolution Algorithms

Rick Boks

Supervisors:
Prof.dr. T.H.W. Bäck and Dr. H. Wang

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

06/08/2019

Abstract

Many variants of Particle Swarm Optimization (PSO) and Differential Evolution (DE) have been proposed, making it difficult to pick a particular algorithm variant for a given problem. Moreover, from an algorithm design perspective, many potentially useful variants still remain unexplored. We introduce a framework in which characteristic components (modules) of existing PSO and DE algorithms are isolated and can be combined into new algorithm instances, many of which have not been considered before. A total of 128 possible algorithm variants resulting from the modular approach are benchmarked on the 24 functions in the COCO framework for different problem dimensions. The instances are compared with regard to their performance on the various function groups and dimensionalities. As a result of the experiments performed by the framework, we discuss combinations of modules that perform well on (specific subsets of) the set of test functions, as well as concrete instances that outperform the original variants.

Contents

Chapter 1

Introduction

In this paper, we consider algorithmic variants of Particle Swarm Optimization (PSO) and Differential Evolution (DE) which aim at solving continuous-variable black-box optimization problems of dimensionality n:

$$f : \mathcal{F} \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$$

where $\mathcal{F} = \prod_{i=1}^{n} [u_i, v_i]$, and without loss of generality a minimization task is assumed, i.e., $f \rightarrow \min$.

These algorithms, which are both inspired by concepts from nature, are two popular techniques to tackle such optimization problems. In recent years, PSO and DE have attracted the attention of many researchers, which has resulted in a large number of variations on the original algorithms. There is no single best algorithm; performance of different algorithms varies greatly when applying them to various optimization problems. For this reason, choosing an algorithm to solve a given problem can be a daunting task. In this work, characteristic operators (modules) of existing PSO and DE variants are combined into hybrid algorithms, many of which have never been considered before. A configurable framework is created, which allows modules that are extracted from existing algorithms to be combined arbitrarily. A similar research in the field of Evolution Strategies has been conducted by van Rijn et al. [vWvB16]. Every hybridized algorithm, which will be referred to as a PSO- or DE instance, is tested on the COCO benchmarking framework [HAM+16], which contains 24 test functions. This work aims at finding hybrid PSO and DE algorithms that outperform the original variants on the set of test functions or on specific subsets thereof. The outline of this paper is as follows: Section 2 introduces the state-of-the-art variants of PSO. Section 3 goes over various leading edge variants of DE. In Section 4, we describe the experimentation that has been performed to benchmark the generated PSO- and DE instances[1]. In Section 5, we define a naming scheme in order to easily refer to all instances. The results of the experiments are then discussed in Section 6. Finally, the conclusion is given in Section 7 and future work is discussed in Section 8.

[1] The source code of the framework is available at: **https://github.com/rickboks/pso-de-framework**.

Chapter 2

Particle Swarm Optimization

Particle Swarm Optimization is an optimization algorithm that mimics the behaviour of a flock of birds, or a school of fish searching for food. PSO was introduced by Eberhart and Kennedy in [EK95].

2.1 The Original Particle Swarm Optimization Algorithm

A particle in a swarm of size $M > 1$ is composed of three vectors: the current position $\mathbf{x_i}$, velocity $\mathbf{v_i}$, and its previous best position $\mathbf{p_i}$, where $i \in \{1, \ldots, M\}$. After the random initialization of $\mathbf{x_i}$ and $\mathbf{v_i}$, the algorithm enters a loop where the $\mathbf{x_i}$ are updated until the termination criteria are met. The update of $\mathbf{x_i}$ is preceded by the update of $\mathbf{v_i}$. $\mathbf{x_i}$ is then calculated by adding $\mathbf{v_i}$ to the current position. The velocity vector is updated using the following equation:

$$\mathbf{v}_i \leftarrow \mathbf{v}_i + \boldsymbol{\mathcal{U}}(0, \phi_1) \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\mathcal{U}}(0, \phi_2) \otimes (\mathbf{p}_g - \mathbf{x}_i) \tag{2.1}$$

where $\boldsymbol{\mathcal{U}}(0, \phi_i)$ represents a vector containing random numbers, uniformly distributed in $[0, \phi_i]$, and $\otimes$ is component-wise multiplication. $\mathbf{p}_g$ represents the best solution that has been found in the neighborhood of particle i. To prevent the velocity from exploding, which can result in particles leaving the search space, $\mathbf{v}_i$ is kept in the range $[-\mathbf{v}_{max}, \mathbf{v}_{max}]$, such that $v_{ij} \in [-V_{max}, V_{max}]$ for a vector $\mathbf{v}_i$ and $j \in \{1, \ldots, n\}$.

After calculating the new velocity, the new position of the particle representing a new candidate solution is calculated as follows:

$$\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i \tag{2.2}$$

After every position update, the current position is evaluated, i.e., its objective function value $f(\mathbf{x}_i)$ is computed. If it is the best solution that has been found by the particle thus far, $\mathbf{p}_i$ is updated with the particle's current position, $\mathbf{x}_i$. The objective function value of $\mathbf{p}_i$ is denoted as $pbest_i = f(\mathbf{p}_i)$.

The pseudo-code of the original PSO algorithm is given in Algorithm 1:

Algorithm 1: Original Particle Swarm Optimization

for *each particle* $\mathbf{x}_i$ **do**
  for $j = 1$ *to* n **do**
    $x_{ij} \leftarrow \mathcal{U}(u_j, v_j)$ ▷ Initialize
    $v_{ij} \leftarrow \mathcal{U}(-V_{\max}, V_{\max})$
  end
  $pbest_i \leftarrow \max$
end
while *termination criteria are not met* **do**
  for *each particle* $\mathbf{x}_i$ **do**
    $f_i \leftarrow f(\mathbf{x}_i)$ ▷ Evaluate
    if $f_i < pbest_i$ **then**
      $\mathbf{p}_i \leftarrow \mathbf{x}_i$ ▷ Select
      $pbest_i \leftarrow f_i$
    end
    Find $\mathbf{x}_i$'s neighbor $\mathbf{p}_g$ with the best value of *pbest*
    $\mathbf{v}_i \leftarrow \mathbf{v}_i + \boldsymbol{\mathcal{U}}(0, \phi_1) \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\mathcal{U}}(0, \phi_2) \otimes (\mathbf{p}_g - \mathbf{x}_i)$
    $\mathbf{x}_i \leftarrow \mathbf{x}_i + \mathbf{v}_i$ ▷ Update velocity and position
  end
end

A termination criterion in PSO can simply be exhausting the maximum amount of function evaluations. Another example is *convergence*, measured by analyzing whether the average distance between particles has gone below a predetermined threshold.

2.2 Velocity Update Strategies

In the original PSO algorithm, the position $\mathbf{x}_i$ of a particle is updated using the following formulas:

$$\begin{aligned} \mathbf{v}_i &\leftarrow \mathbf{v}_i + \boldsymbol{\mathcal{U}}(0, \phi_1) \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\mathcal{U}}(0, \phi_2) \otimes (\mathbf{p}_g - \mathbf{x}_i) \\ \mathbf{x}_i &\leftarrow \mathbf{x}_i + \mathbf{v}_i \end{aligned} \tag{2.3}$$

In this original approach, it is important that the value for $\mathbf{v}_{max}$ is chosen carefully. This is, however, not a trivial task since the optimal value is problem-specific. This is not the only problem that using hard velocity bounds introduces. Particles often fail to converge when using $[-\mathbf{v}_{max}, \mathbf{v}_{max}]$ because this approach fails to move from an exploratory search to an exploitative one. In an attempt to move away from using hard velocity bounds, a modified version of the update function was introduced in [SE98]. An *inertia weight* ω is used in order to better control the scope of the search without the use of V_{max}:

$$\begin{aligned} \mathbf{v}_i &\leftarrow \omega \mathbf{v}_i + \boldsymbol{\mathcal{U}}(0, \phi_1) \otimes (\mathbf{p}_i - \mathbf{x}_i) + \boldsymbol{\mathcal{U}}(0, \phi_2) \otimes (\mathbf{p}_g - \mathbf{x}_i) \\ \mathbf{x}_i &\leftarrow \mathbf{x}_i + \mathbf{v}_i \end{aligned} \tag{2.4}$$

A large value of ω will result in an exploratory search, while a small value corresponds to a more exploitative search. Because it is desirable to move from exploratory to exploitative search, it is no surprise that researchers found that the inertia weight performs especially well when decreasing the value of ω over time. Both the

original inertia weight and the version with the decreasing value of ω are considered in this paper. When using the decreasing inertia weight, ω is decreased from 0.9 to 0.4 throughout the run of the algorithm. A value $\phi_1 = \phi_2 = 1.49618$ is chosen during this research according to a recommendation given in [CK02].

Instead of only being influenced by the best neighbor, the velocity of a particle in the Fully Informed Particle Swarm (FIPS) [KM02a] is updated using the best previous positions of *all* its neighbors. The corresponding equation is:

$$\mathbf{v}_i \leftarrow \chi\Big(\mathbf{v}_i + \frac{1}{K_i}\sum_{j=1}^{K_i} \boldsymbol{\mathcal{U}}(0,\phi) \otimes (\mathbf{p}_{nbr_j} - \mathbf{x}_i)\Big) \tag{2.5}$$

where K_i is the number of neighbors of particle i and nbr_j is the jth neighbor of particle i. The value of χ is computed as

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}}\,. \tag{2.6}$$

The value $\phi = 4.1$ is chosen according to the recommendation by the authors, which results in a value $\chi = 0.7298$.

Finally, Bare-Bones PSO is a completely different approach from those listed previously. In fact, it uses no velocity at all. In Bare-Bones PSO, every component x_{ij} of $\mathbf{x}_i$ is updated according to a Gaussian probability distribution with mean $\frac{p_{ij}+g_{ij}}{2}$ and variance $|p_{ij} - g_{ij}|$ instead, where p_{ij} and g_{ij} are the jth component of $\mathbf{p}_i$ and $\mathbf{g}_i$, respectively:

$$x_{ij} \sim \mathcal{N}\left(\tfrac{p_{ij}+g_{ij}}{2}, |p_{ij} - g_{ij}|\right) \tag{2.7}$$

2.3 Population Topologies

Eight different topologies from the literature have been implemented in the framework, including both static and dynamic topologies. Static topologies remain the same throughout the entire run, while dynamic topologies can add and remove connections between particles as the run progresses.

In the *lbest* (local best) or ring topology as proposed in [EK95], the target particle is only influenced by its two adjacent neighbors in the array. In this topology, information about newly found solutions travels relatively slowly through the population. This gives the *lbest* topology an explorative property, which reduces the risk of getting 'stuck' in local optima, but also reduces the convergence rate of the swarm.

The *gbest* (global best) [EK95] or star topology is implemented as a fully connected graph, so that every particle is influenced by the best solution found in the *entire swarm*. It converges faster than the *lbest* topology, but performs a less explorative search. The *gbest* topology is more susceptible to being attracted by local optima.

Multiple topologies were experimented with in [Ken99], two of which will be considered in this paper: the random graph and the wheel topology. In the random graph topology, every particle is connected to three randomly selected neighbors from the topology. These connections are, in contrast to most other topologies, *unidirectional*: If, for particles i and j, $i \in neighborhood(j)$, it does not necessarily mean that $j \in neighborhood(i)$.

The wheel topology is implemented as follows: *one* particle is connected to all the others, and all the other particles are only connected to this central particle.

Among others, the Von Neumann topology was tested by Kennedy and Mendes in [KM02b]. This topology is a good trade-off between the *lbest* and *gbest* topologies. Particles are arranged in a two-dimensional array, and have four neighbors: the ones horizontally and vertically adjacent to them, with toroidal wrapping. This way, it allows for the parallel search that the *lbest* topology excels at, while at the same time retaining good convergence properties. The Von Neumann topology in particular performed very well in the experiments performed by Kennedy and Mendes.

The final three topologies are dynamic: In contrast to the static topologies, their connections change throughout the run of the algorithm.

The increasing topology was proposed in [Sug99]. The idea behind this topology is to utilize the best of both the *lbest* and *gbest* topologies by starting with an *lbest* topology and gradually increasing the connectivity so that, by the end of the run, the particles are fully connected. The connectivity C_i of particle i (i.e. the number of neighbors particle i has), at any time is determined by:

$$C_i = 2 + [C_{max} \cdot (\#evals/evalBudget)] \tag{2.8}$$

Here, C_{max} is the maximum connectivity, which in our case is $M - 1$, where M is the population size. $\#evals$ and $evalBudget$ denote number of evaluations performed and the maximum number of evaluations, respectively. New connections are chosen randomly.

The counterpart of the increasing topology, though potentially very interesting, has never been considered before. The implementation of this *decreasing* topology is analog to the increasing one: We start off with a fully connected swarm, and remove randomly chosen connections between particles as the run progresses until the particles are connected with a ring topology by the end of the run. The connectivity of every particle at any time is determined by:

$$C_i = C_{max} - [(C_{max} - 2) \cdot (\#evals/evalBudget)] \tag{2.9}$$

In all previously discussed topologies, the particles are connected in a single cluster. The dynamic multiswarm topology (DMS-PSO), as proposed in [LS05], creates clusters consisting of three particles each, and creates new clusters after every 5 iterations. If the population size is not divisible by three, every cluster has size three, except one, which is of size $3 + (M \mod 3)$.

Chapter 3

Differential Evolution

Differential Evolution (DE) was first introduced by Storn and Price in 1995 [SP95]. This original version already proved its effectiveness at various contests and conferences, and has been improved since, resulting in many different variants of the algorithm. Differential Evolution works in a similar fashion as other evolutionary algorithms. However, it uses scaled differences of randomly selected members of the population in order to perturb members of the current population, called *genomes*. We adopt this original DE teminology here, although the genomes could also be called particles or individuals. The algorithm consists of four main parts: initialization, mutation, crossover and selection.

3.1 The Original Differential Evolution Algorithm

The first step of the algorithm is the initialization of the genomes. The first generation of genomes should cover the search space as uniformly as possible. Genomes are initialized uniformly at random inside the bounds of the search space.

After initialization, the main loop of the DE algorithm commences. In this loop, the genomes of the current population (the target vectors) are first mutated, which creates perturbations of the target vector with a random element. These mutated vectors are called *donor vectors*. After mutation, the previously created donor vectors are combined with the target vectors in order to enhance the diversity of the population. The vectors that are generated in this crossover step are called *trial vectors*. Finally there is the selection step. In this step, the best genomes from the union of the target- and trial vectors are selected for the next generation.

Mutation of a single genome in DE requires three other genomes from the current population. These vectors are selected uniformly at random and will be referred to as $\mathbf{x}_{r_1^i}$, $\mathbf{x}_{r_2^i}$, and $\mathbf{x}_{r_3^i}$, where the indices $r_1^i \neq r_2^i \neq r_3^i \neq i$. The donor vector $\mathbf{v}_i$ is created by adding the scaled difference of two of the randomly-selected vectors to the third one.

$$\mathbf{v}_i \leftarrow \mathbf{x}_{r_1^i} + F \cdot (\mathbf{x}_{r_2^i} - \mathbf{x}_{r_3^i}) \tag{3.1}$$

Here, F is a scalar value called the *mutation rate*, which is often chosen in the interval $[0.4, 1]$.

After the generation of the donor vectors, the trial vectors are created through crossover. The trial vector is created by taking some elements from the target vector and some from the donor vector. Two different crossover methods can be used: exponential- and binomial crossover. In exponential crossover, two integers p, $q \in \{1, \ldots, n\}$ are chosen. The integer p acts as the starting point where the exchange of components begins, and is chosen randomly. q represents the number of elements that will be given by the donor vector, and is chosen using Algorithm 2.

Algorithm 2: Algorithm for assigning a value to q

```
q ← 0
do
    q ← q + 1;
while ((rand(0, 1) ≤ Cr) and (q ≤ n))
```

In the pseudo-code above, Cr is the *crossover rate*, which determines how much influence the donor vector has in the crossover process. The values p and q are re-determined for every genome in the population, every iteration. The trial vector is generated as:

$$u_{ij} \leftarrow \begin{cases} v_{ij} \text{ for } j = \langle p \rangle_n, \langle p+1 \rangle_n \ldots \langle p+q-1 \rangle_n \\ x_{ij} \text{ for all other } j \in \{1, \ldots, n\} \end{cases} \tag{3.2}$$

The angular brackets $\langle\rangle_n$ denote the modulo operator with modulus n. u_{ij} is the jth element of trial vector $\mathbf{u}_i$, v_{ij} the jth element of donor vector $\mathbf{v}_i$, etc.

Binomial crossover is a much simpler approach, yet at the same time the more popular crossover method of the two, because it seems to be the most competitive on most test functions. Binomial crossover works as follows: a random number $k \sim \mathcal{U}(0, 1)$ is chosen for every element of the trial vector. If the jth random number k_j is less or equal to the crossover rate Cr, the jth element of the trial vector is inherited from the jth element of the donor vector, else it is assigned the jth element of the target vector. In order to make sure that at least one element of the donor vector is inherited by the trial vector, a random number $j_{rand} \in [1, D]$ is chosen. The j_{rand}th item of the donor vector will always be inherited by the trial vector.

The value of j_{rand} is re-determined for each vector, every iteration. Binomial crossover can be outlined as:

$$u_{ij} \leftarrow \begin{cases} v_{ij} \text{ if } rand(0, 1) \leq Cr \text{ or } j = j_{rand} \\ x_{ij} \text{ otherwise} \end{cases} \tag{3.3}$$

The final step in the main loop of the DE algorithm is selection. Every iteration, new trial vectors are generated during the crossover step. We can, however, not just keep adding these newly generated trial vectors to the population; the population size needs to remain constant. For this reason, only the best individual from every

target- and trial vector pair is kept for the next generation:

$$\mathbf{x}_i \leftarrow \begin{cases} \mathbf{u}_i \text{ if } f(\mathbf{u}_i) < f(\mathbf{x}_i) \\ \mathbf{x}_i \text{ otherwise} \end{cases} \tag{3.4}$$

The termination criteria in DE usually take the same form as for PSO. To give a clear overview of the complete Differential Evolution algorithm, the pseudo-code is provided in Algorithm 3.

Algorithm 3: Differential Evolution using binomial crossover

for *each genome* $\mathbf{x}_i$ **do**
  for $j = 1$ *to* n **do**
    $x_{ij} \leftarrow \mathcal{U}(u_j, v_j)$ ▷ Initialize
  end
end
while *termination criteria are not met* **do**
  for *each genome* $\mathbf{x}_i$ **do**
    $\mathbf{v}_i \leftarrow \mathbf{x}_{r_1^i} + F \cdot (\mathbf{x}_{r_2^i} - \mathbf{x}_{r_3^i})$ ▷ Mutate
  end
  for *each donor vector* $\mathbf{u}_i$ **do**
    for $j = 1$ *to* n **do**
      $p \leftarrow \mathcal{U}(\{1, \ldots, n\})$
      Compute q according to Algorithm 2
      if $j \in \{\langle p \rangle_n, \langle p+1 \rangle_n \ldots \langle p+q-1 \rangle_n\}$ **then**
        $u_{ij} \leftarrow v_{ij}$ ▷ Crossover
      else
        $u_{ij} \leftarrow x_{ij}$
      end
    end
  end
  for $i = 1$ *to* M **do**
    if $f(\mathbf{u}_i) < f(\mathbf{x}_i)$ **then**
      $\mathbf{x}_i \leftarrow \mathbf{u}_i$ ▷ Select
  end
end

3.2 Population Initialization Strategies

In original DE, genomes are initialized randomly within the bounds of the search space. This is generally a decent technique, but can be improved by also taking a look at the so-called *opposite* of the random guesses. Opposition-based population initialization is part of Opposition-Based Differential Evolution, as proposed by Rahnamayan *et al.* [RTS08]. Opposition-Based Differential Evolution modifies the original DE algorithm at three different stages. Apart from opposition-based population initialization, there is opposition-based generation jumping, which will be discussed in section 3.4, and opposition-based individual jumping, where the best genome from the population can potentially be replaced by an improved version. This last concept is not considered in this research.

Opposition-based population initialization initially creates a randomly initialized population P, just like in the original approach. Then the opposite of P, O is created. The elements of O are constructed as follows:

$$o_{ij} \leftarrow u_j + v_j - p_{ij} \tag{3.5}$$

Here, p_{ij} is the jth component of the ith genome from the population P (analog for o_{ij}). u_j and v_j are the minimum and maximum bounds of the search space for the jth solution vector component, respectively. When O is constructed, the best M genomes from the set $P \cup O$ form the first generation. Because the algorithm starts off with the better of the two guesses, it will likely converge faster.

3.3 Mutation Techniques

The original mutation scheme, as mentioned in section 3.1, uses three randomly selected, mutually exclusive vectors to create its donor vectors. One of the vectors is used as the base vector. In the original approach, only one weighted difference vector, $F \cdot (\mathbf{x}_{r_2^i} - \mathbf{x}_{r_3^i})$, is used. For this reason, this mutation scheme is called DE/rand/1. The 'rand' refers to the base vector that is randomly selected, and the '1' refers to the single difference vector that is used. Storn and Price [SP95] proposed four other mutation schemes, which are named using the same naming convention.

The first one is DE/best/1. It is similar to DE/rand/1, except that DE/best/1 uses the vector with the best objective function value instead of a randomly selected one as the base vector:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{best} + F \cdot (\mathbf{x}_{r_1^i} - \mathbf{x}_{r_2^i}) \tag{3.6}$$

DE/target-to-best/1 uses the target vector as the base vector, and also uses the weighted difference between the best- and target vector:

$$\mathbf{v}_i \leftarrow \mathbf{x}_i + F \cdot (\mathbf{x}_{best} - \mathbf{x}_i) + F \cdot (\mathbf{x}_{r_1^i} - \mathbf{x}_{r_2^i}) \tag{3.7}$$

DE/best/2 uses two scaled differences of randomly chosen vectors:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{best} + F \cdot (\mathbf{x}_{r_1^i} - \mathbf{x}_{r_2^i}) + F \cdot (\mathbf{x}_{r_3^i} - \mathbf{x}_{r_4^i}) \tag{3.8}$$

Finally, similar to DE/best/2, DE/rand/2 uses two scaled differences of randomly chosen vectors, but also uses a randomly chosen vector as base:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{r_1^i} + F \cdot (\mathbf{x}_{r_2^i} - \mathbf{x}_{r_3^i}) + F \cdot (\mathbf{x}_{r_4^i} - \mathbf{x}_{r_5^i}) \tag{3.9}$$

Apart from the mutation techniques from Storn and Price's DE family, three others will be considered. The first one, DE/rand/2/dir, which was proposed in [FJ04], can be described as follows:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{r_1{}^i} + \frac{F}{2} \cdot (\mathbf{x}_{r_1^i} - \mathbf{x}_{r_2^i} + \mathbf{x}_{r_3^i} - \mathbf{x}_{r_4^i}) \tag{3.10}$$

where $\mathbf{x}^i_{r_1}$, $\mathbf{x}^i_{r_2}$, $\mathbf{x}^i_{r_3}$ and $\mathbf{x}^i_{r_4}$ are four distinct population members such that $f(\mathbf{x}^i_{r_1}) \leq f(\mathbf{x}^i_{r_2})$ and $f(\mathbf{x}^i_{r_3}) \leq f(\mathbf{x}^i_{r_4})$.

Das *et al.* proposed a new neighborhood-based mutation scheme [DACK09] in a desire to improve DE/target-to-best/1, which performs poorly on multimodal fitness landscapes. The resulting algorithm, called DEGL, uses neighborhoods similar to PSO in order to prevent the premature convergence which DE/target-to-best/1 displays because the entire population is attracted by the same best found solution. In DEGL, a radius k is chosen, which determines the size of every particle's neighborhood $(2k)$. The neighborhood of every vector $\mathbf{x}_i$ in the population includes the k vectors to the left and to the right of it in the array, such that $\mathbf{x}_{i-1}$ and $\mathbf{x}_{i+1}$ are $\mathbf{x}_i$'s direct neighbors. A local and a global vector are created in order to compose the donor vector. The local vector is created as follows:

$$\boldsymbol{\ell}_i \leftarrow \mathbf{x}_i + \alpha \cdot (\mathbf{x}_{n\_best_i} - \mathbf{x}_i) + \beta \cdot (\mathbf{x}_p - \mathbf{x}_q) \tag{3.11}$$

where n_best_i is the index of the vector in the neighborhood of $\mathbf{x}_i$ with the best objective function value, p and q are indices of two random vectors in the neighborhood of $\mathbf{x}_i$, and α and β are scaling factors. The global vector is created similarly:

$$\mathbf{g}_i \leftarrow \mathbf{x}_i + \alpha \cdot (\mathbf{x}_{g\_best} - \mathbf{x}_i) + \beta \cdot (\mathbf{x}_{r1} - \mathbf{x}_{r2}) \tag{3.12}$$

where g_best is the index of the best vector in the entire population, and r_1, r_2 are two random indices with $r_1 \neq r_2 \neq i$. Finally, the donor vector is created by combining the local and global vector:

$$\mathbf{v}_i \leftarrow w_i \cdot \mathbf{g}_i + (1 - w_i) \cdot \boldsymbol{\ell}_i \tag{3.13}$$

where w_i is a self-adaptive scalar weight in $(0, 1)$, which is updated for every vector, in every generation. The update procedure is outlined in [DACK09]. In this work, the values of α and β are chosen according to recommendations of the authors: $\alpha = \beta = F$. A value of neighborhood radius $k = 3$ is used as a result of our previous experimentation.

The final mutation technique that will be considered in this paper, known as NSDE, which was introduced by Yang *et al.* [YYH07], performs mutation using:

$$\mathbf{v}_i \leftarrow \mathbf{x}_{r_1^i} + \begin{cases} \mathbf{d}_i \cdot N(0.5, 0.5) \text{ if } rand(0,1) < 0.5 \\ \mathbf{d}_i \cdot \delta \text{ otherwise} \end{cases} \tag{3.14}$$

where $\mathbf{d}_i = \mathbf{x}_{r_2^i} - \mathbf{x}_{r_3^i}$, N is a normal distribution, and δ is a Cauchy random variable with location parameter $t = 0$ and scale parameter $s = 1$.

3.4 Opposition-based Generation Jumping

Opposition-based generation jumping is one of three components of Opposition-Based Differential Evolution [RTS08]. It forces the evolutionary process to jump to a fitter generation. After the usual process in each iteration of mutation, crossover and selection, there is a chance Jr that opposition-based generation jumping occurs. When this happens, the opposite of the current population, OP, is calculated and the M fittest vectors from the set $P \cup OP$ are selected. It is important to note that the minimum and maximum values of every variable *in the current population* are used instead of the predefined a_j and b_j. If we would use the static boundaries, like in opposition-based population initialization, we would lose progress when vectors jump out of the search space that has shrunken during the progression of the run. For Jr, the authors recommend a value of 0.3 which will be used in this work.

Chapter 4

Experimental Setup

A framework has been implemented to create PSO- and DE instances out of arbitrary combinations of modules. The framework is written in C++ and uses the COCO (COmparing Continuous Optimizers) benchmarking framework [HAM+16] to measure the performance of the created algorithms. Various topologies, velocity update functions, mutation strategies, etc. have been implemented in the framework, including the ones that are experimented with in this work. A suite of PSO or DE instances can be instantiated, that contains every possible instance that can be created with the implemented modules, or with just a desired subset of modules. It is possible to create all possible combinations of operators and experiment with the resulting algorithm. The structure of the framework allows for easy implementation of extensions, such as new PSO topologies, velocity update functions, DE initialization strategies, mutation techniques, etc.

In the experiments conducted, a PSO instance is seen as a combination of three modules: the population topology, velocity update technique and the "synchronicity" of the updates of the $pbest_i$'s and $gbest$'s (either synchronous or asynchronous). The eight population topologies discussed in section 2.3 have been implemented in the framework, as well as the four different velocity update strategies as mentioned in section 2.2. Velocity update strategies and population topologies can be combined arbitrarily in a PSO instance, and every PSO instance can have synchronous or asynchronous updates. Combining every velocity update strategy, population topology and synchronicity results in a total of $8 \cdot 4 \cdot 2 = 64$ different PSO instances.

A DE instance is composed of four modules: the initialization strategy, mutation technique, crossover technique, and opposition-based generation jumping, which can be either enabled or disabled. The initialization technique can be one of the two that were discussed in section 3.2. The second module, mutation, can be realized by one of the eight different techniques described in section 3.3. One of the two crossover techniques described in section 3 can be used for the third module. By combining all possible modules we also obtain $2 \cdot 8 \cdot 2 \cdot 2 = 64$ different DE instances.

All instances are tested on the COCO benchmarking framework [HAM+16], containing 24 test functions. Every PSO- and DE instance is given a budget of $10^4 D$ function evaluations. As a result of previous experimentation, a population size of $6D$ and $8D$ are used for PSO and DE, respectively. Furthermore, the mutation rate F has

value 0.5 and crossover rate $Cr = 0.7$ for DE. To get a reliable result, every instance is run on the first function instance of every function for a total of 50 times.

The experiment is run on the DAS-5 cluster, which allows all 64 variants to be run in parallel, using MPI. COCO produces data of the performance of every PSO- and DE instance on every function, on every dimension. As a measure of performance, the average runtime (aRT) is used, which is the total number of function evaluations an algorithm used to reach a certain target during all of its runs (in our case 50), divided by the number of successful runs. The measured aRT's of the best performing algorithms on every test function during the Black-Box Optimization Benchmarking workshop in 2009 are also provided by COCO, which are used as a reference. After post-processing the data with the tool provided by COCO, we obtain tables for every function on every dimension, containing the aRT's of every PSO- and DE instance *divided by the aRT of the reference algorithm*. In order to determine which instance performs best on a given function group and dimension, a ranking is created for every function group, for every dimension. This is done by sorting the instances on their aRT ratios on every one of the seven targets. If the target was never reached, the instance will always get the lowest possible rank: 64. If two instances have the same aRT ratio, they will get the same rank. Then, the 'average rank' of every instance on this function and dimension is calculated by averaging the seven ranks. In order to get a more easily digestible result, we calculate the average rank of every instance on every *function group*, instead of on individual functions. This is simply done by taking the average of the members of every function group. The function groups will be referred to by using a value $1 \ldots 5$. The corresponding function group descriptions are provided in table 4.1.

#	Description
1	Seperable functions
2	Functions with low or moderate conditioning
3	Unimodal functions with high conditioning
4	Multi-modal functions with adequate global structure
5	Multi-modal functions with weak global structure

Table 4.1: COCO function group descriptions

Chapter 5

Naming Convention of Algorithms

In order to easily refer to any PSO- and DE instance, a naming scheme is introduced. Because any instance can be characterized by the modules it is composed of, it will be named using abbreviations of the modules. A PSO instance is named using the following scheme: $PSO\_X\_Y\_Z$, where X represents the velocity update strategy, Y denotes the topology and Z is either S or A, which represent synchronous and asynchronous updates, respectively. Tables 5.1 and 5.2 show the abbreviations for all velocity update strategies and topologies.

DE instances are named using a similar scheme: it can be described by $DE\_W\_X\_Y\_Z$, where W denotes the initialization strategy, X is the mutation technique, Y the crossover technique, and Z represents opposition-based generation jumping. The initialization technique W is either O (opposition-based) or R (random). The Y can be substituted by a B if binomial crossover is used, and E in the case of exponential crossover. The value for Z is either 0 or 1, which shows whether opposition-based generation jumping is enabled or disabled. Lastly, the mutation technique X can be one of the eight described in section 3.3. The eight corresponding abbreviations can be found in table 5.3.

X	Velocity update
B	Bare-Bones PSO (BPSO)
F	Fully-informed PSO (FIPS)
I	Inertia weight
D	Decreasing inertia weight

Table 5.1: Velocity update encodings

Y	Topology
L	*lbest* (ring)
G	*gbest* (fully connected)
R	Random graph
N	Von Neumann
W	Wheel
I	Increasing connectivity
D	Decreasing connectivity
M	Dynamic multi-swarm

Table 5.2: Population topology encodings

X	Mutation
$R1$	DE/rand/1
$B1$	DE/best/1
$TTB1$	DE/target-to-best/1
$B2$	DE/best/2
$R2$	DE/rand/2
$R2D$	DE/rand/2/dir
TOP	Topology-based (DEGL)
NS	NSDE

Table 5.3: Mutation encodings

Chapter 6

Results

Table 6.1 shows the PSO instances that performed best in the conducted experiments on the various dimensions and function groups. It is important to note that these results do not necessarily prove the superior performance of one algorithm instance over another. The number of function evaluations and runs that every algorithm configuration was given on each of the function instances is not large enough to facilitate a decisive conclusion about the relative performance of the algorithms. In many cases, the best-performing instance on some dimension and function group is closely followed by an instance which only performed slightly worse. In these cases, the ranking based on the performance of the instances could partly be a result of the random element of both PSO and DE. It is possible that different results are obtained when running the instances for a larger number of times on every problem.

#	2-D	5-D	20-D
1	*PS_B_G_S*	*PS_B_D_S*	*PS_B_G_S*
2	*PS_I_N_S*	*PS_F_R_S*	*PS_F_M_S*
3	*PS_I_N_A*	*PS_F_N_A*	*PS_F_R_S*
4	*PS_B_N_S*	*PS_F_R_S*	*PS_F_R_S*
5	*PS_I_M_A*	*PS_I_M_A*	*PS_F_L_S*

Table 6.1: Best performing PSO instances on all function groups and dimensions

A very apparent observation is that Bare-Bones PSO performed significantly better than any other velocity update (in BPSO's case position update) strategy on the first function group. In all dimensions, the top 7 to 15 best performing algorithms use BPSO. Out of the topologies we considered, BPSO shows the best performance in combination with the *gbest* and the decreasing connectivity topology. In 2 dimensions, BPSO is also the best performing velocity update strategy on function group 4. On the other three function groups, the inertia weight is the best choice, particularly when combined with the Von Neumann- or dynamic multi-swarm topology.

When looking at 5 dimensions, we see that the Fully-Informed Particle Swarm performs well on nearly all function groups, namely 2, 3, 4, and 5. The best topologies to use in combination with FIPS are Von Neumann, *lbest* and the random topology. The performance of the inertia weight in 5-D is comparable to that of FIPS on

function groups 2, 3, and 5 when using it with the increasing connectivity-, dynamic multi-swarm-, or the Von Neumann topology.

In 20-D, FIPS remains a good, if not the best choice for every function group except the first. On function groups 2, 3, and 4, FIPS is best combined with the random, dynamic multi-swarm, or von Neumann topology. However, FIPS performed best on function group 5 when it was used with the *lbest* topology. The decreasing inertia weight seems to perform relatively well on the fifth function group, in all three dimensionalities, when combined with the Von Neumann- or multi-swarm topology.

In general, the best choice of using synchronous- or asynchronous updates appears to depend on the topology that is used. For example, the *gbest* topology favors synchronous updates, and instances with the *lbest* topology generally work better with asynchronous updates. There does, however not seem to be a rule that decides whether to choose for synchronous or asynchronous updates, for any topology.

It is clear that the PSO algorithm is very sensitive to the choice of modules. An instance that shows poor performance on one function group can perform great on another. The optimal combination of modules also varies between dimensionalities, which further complicates choosing a PSO instance for a given function group.

A similar summary of the best performing DE instances is shown in table 6.2. The best/2 scheme consistently performs well in two dimensions, especially on function groups 1,2 and 3. On function group 4 and 5, instances using target-to-best/1 or rand/1 mutation also showed good performance. The mutation scheme appears to be the only module with significant influence on the performance of a DE instance in 2-D. When an instance with a certain mutation scheme performs well, most or all of the other instances with this mutation scheme also perform well. Opposition-based initialization and -generation jumping do, however seem to improve the performance of poorly performing instances to some extent.

#	2-D	5-D	20-D
1	$DE\_R\_B2\_BIN\_0$	$DE\_R\_B1\_EXP\_1$	$DE\_R\_B1\_EXP\_1$
2	$DE\_O\_B2\_EXP\_0$	$DE\_R\_B1\_EXP\_0$	$DE\_O\_B2\_BIN\_1$
3	$DE\_O\_B2\_BIN\_1$	$DE\_R\_TTB1\_BIN\_0$	$DE\_R\_B1\_BIN\_1$
4	$DE\_R\_R1\_EXP\_0$	$DE\_O\_NS\_BIN\_1$	$DE\_R\_TTB1\_BIN\_1$
5	$DE\_O\_B2\_BIN\_1$	$DE\_R\_B1\_BIN\_1$	$DE\_O\_TTB1\_BIN\_0$

Table 6.2: Best performing DE instances on all function groups and dimensions

In 5 dimensions, the best/2 scheme is no longer most successful. The best/1 scheme is part of successful instances on all function groups. The performance of best/1 is only topped on function groups 3 and 4, by target-to-best/1 and the NSDE mutation scheme, respectively. Surprisingly, $DE\_O\_NS\_BIN\_1$ was the best performing instance on function group 4, while no other instances using NSDE made it to the top 10. Function groups 1 and 4 show a clear preference regarding the crossover scheme. Exponential crossover shows much better performance on function group 1, while the better crossover scheme for function group 4 is clearly binomial.

In 20 dimensions, this pattern becomes even more clear, now for all function groups. The preferred crossover schemes for function groups 1 and 4 remain the same. For the other function groups, binomial crossover is the most competitive choice. Choosing the right crossover scheme seems even more crucial than the mutation scheme, in 20-D. Best/1 again performed well on all function groups. Target-to-best/1 also showed good performance compared to others, except on function group 1. The best/2 scheme was used in the best instances on function group 2, specifically in combination with opposition-based generation jumping. In general, opposition-based generation jumping seems to be beneficial to most instances, but the difference in performance is generally not large. The effect of opposition-based initialization on the performance of an instance does not show in 20-D.

Similar to PSO, choosing the right combination of modules for a DE instance for a certain job is vital. The performance differences between DE instances are, however, generally smaller than those of PSO. The performance of the best PSO instance on a certain function group and dimension is often similar to that of the best DE instance, but on average DE performs much better than PSO.

ECDF graphs of some of the best-performing instances are provided in Appendix A. Tables with the aRT's of all instances in 2-D and 5-D on a fixed target are available in Appendix B.

Chapter 7

Conclusion

A framework has been created in which characteristic parts (modules) of existing Particle Swarm Optimization- and Differential Evolution variants are extracted and combined into new instances. Modules can be combined arbitrarily in the framework, which allows experimentation with unorthodox instances that have not been considered before. In this work, 64 PSO instances and 64 DE instances have been generated by the framework and have been tested on the 24 test functions provided by the COCO benchmarking framework. By introducing a method to rank the instances with regard to their performance, we obtain a ranking of all the instances on various function groups and dimensions. Various interesting PSO- and DE instances have been found that outperform the classical variants on given function groups and dimensions. As to be expected, no single instance performs best on all functions and dimensions. However, we can conclude that some combinations of modules form excellent instances that outperform their original counterparts.

To give an example, the Fully Informed Particle Swarm (FIPS) showed great performance on a subset of the test functions, when used in conjunction with the dynamic multi-swarm topology and the topology with random connections between particles. Bare-Bones PSO combined with a decreasing connectivity topology also showed good performance. More specific instances that performed well can be named, as well as individual or pairs of modules that have proven to be successful for certain jobs.

We observed large performance differences between various DE mutation- and crossover schemes on specific function groups and dimensionalities. It is important to carefully consider which operators are used when choosing a DE instance because the optimal algorithm is strongly dependant on the problem at hand. In particular, the choice of crossover scheme becomes very important as the dimensionality grows. Another interesting observation is that opposition-based generation jumping generally contributes to a better performance of a DE instance.

Differential Evolution seems to be the more competitive optimization algorithm of the two. Although the best PSO instance on a given problem generally performs similarly to the best performing DE instance, PSO has much larger differences in performance which makes choosing the optimal algorithm a lot more difficult.

Chapter 8

Future Work

Only a limited amount of modules have been experimented with in this work, in order to restrict the resulting PSO- and DE instances to a reasonable number. The benefit of 'only' considering 128 instances is that we can give every instance a relatively large objective function evaluation budget and many runs on every problem, to get a more reliable result. The obvious downside is that many topologies, velocity update strategies, mutation techniques and crossover techniques are left unconsidered.

Moreover, an interesting extension of this approach is to combine all modules from PSO and from DE and to allow for arbitrary combinations of their modules, thereby increasing the design space significantly. Similar to the combinatorial design space that was introduced by van Rijn *et al.* for evolution strategies [vWvB16], we can then generate and compare thousands of new algorithms or even use a genetic algorithm to search this algorithm design space.

Bibliography

[CK02] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, Feb 2002.

[DACK09] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar. Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, June 2009.

[EK95] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. *Proceedings of the sixth international symposium on micro machine and human science*, pages 39–43, 1995.

[FJ04] V. Feoktistov and S. Janaqi. Generalization of the strategies in differential evolution. In *18th International Parallel and Distributed Processing Symposium, 2004. Proceedings.*, pages 165–, April 2004.

[HAM+16] N. Hansen, A. Auger, O. Mersmann, T. Tušar, and D. Brockhoff. COCO: A platform for comparing continuous optimizers in a black-box setting. *ArXiv e-prints*, arXiv:1603.08785, 2016.

[Ken99] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1931–1938 Vol. 3, July 1999.

[KM02a] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1671–1676 vol.2, May 2002.

[KM02b] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, volume 2, pages 1671–1676 vol.2, May 2002.

[LS05] J. J. Liang and P. N. Suganthan. Dynamic multi-swarm particle swarm optimizer. In *Proceedings 2005 IEEE Swarm Intelligence Symposium, 2005. SIS 2005.*, pages 124–129, June 2005.

[RTS08] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama. Opposition-based differential evolution. *IEEE Transactions on Evolutionary Computation*, 12(1):64–79, Feb 2008.

[SE98] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, May 1998.

[SP95] Rainer Storn and Kenneth Price. Differential evolution: A simple and efficient adaptive scheme for global optimization over continuous spaces. *Journal of Global Optimization*, 23, 01 1995.

[Sug99] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, volume 3, pages 1958–1962 Vol. 3, July 1999.

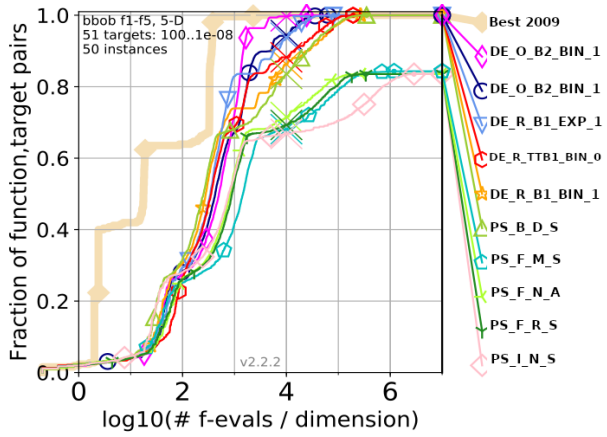
[vWvB16] S. van Rijn, H. Wang, M. van Leeuwen, and T. Bck. Evolving the structure of evolution strategies. In *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 1–8, Dec 2016.

[YYH07] Zhenyu Yang, Xin Yao, and Jingsong He. *Making a Difference to Differential Evolution*, pages 397–414. Springer Berlin Heidelberg, 12 2007.

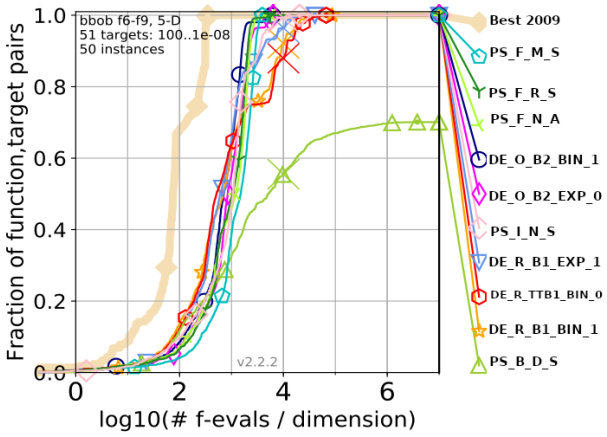
Appendix A

ECDF Graphs

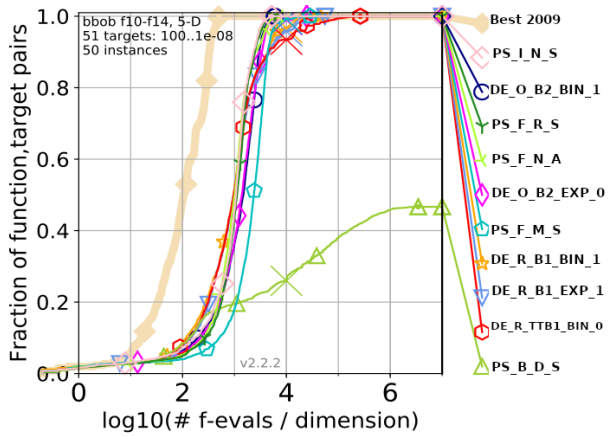
Below are the ECDF graphs of 5 of the best PSO instances and 5 of the best DE instances, on all 5 function groups in 5 dimensions and 20 dimensions. The best algorithm from the Black-Box Optimization Benchmarking workshop in 2009 is highlighted as a thick yellow line as a reference. The ECDF graphs in 2 dimensions are omitted because they are unclear and not as informative due to the smaller performance differences. We recommend Table B.1 and Table B.2 as a reference for the instances in 2 dimensions instead.



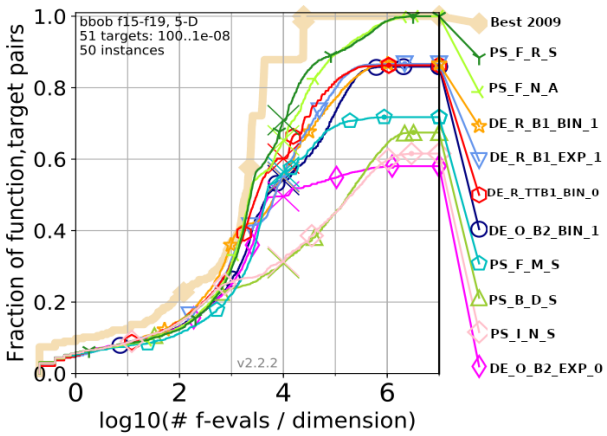
Function group 1, 5-*D*



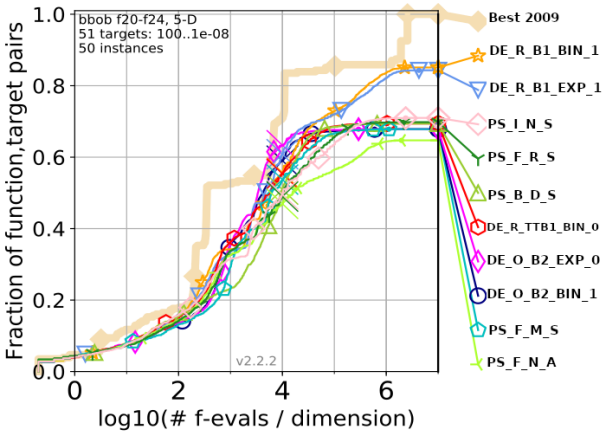
Function group 2, 5-*D*



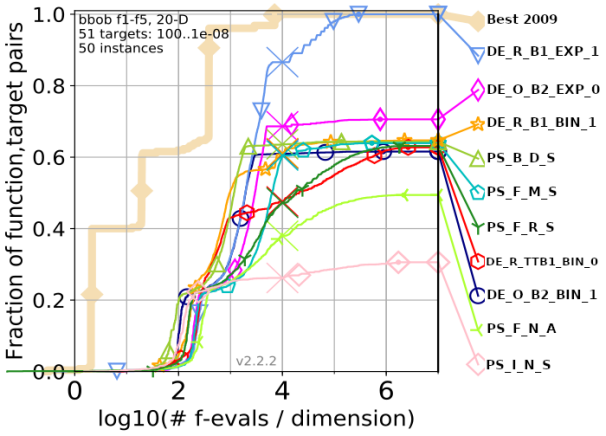
Function group 3,5-*D*



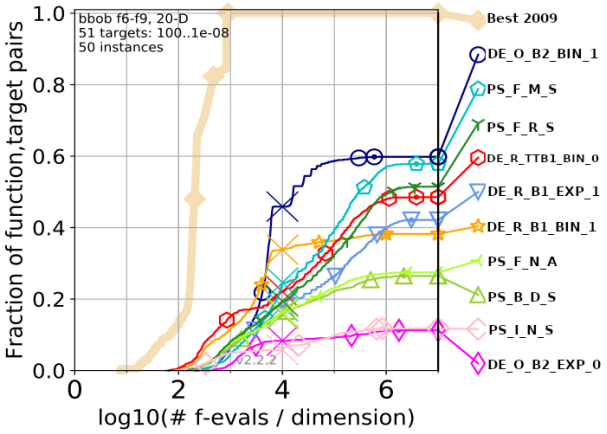
Function group 4, 5-*D*



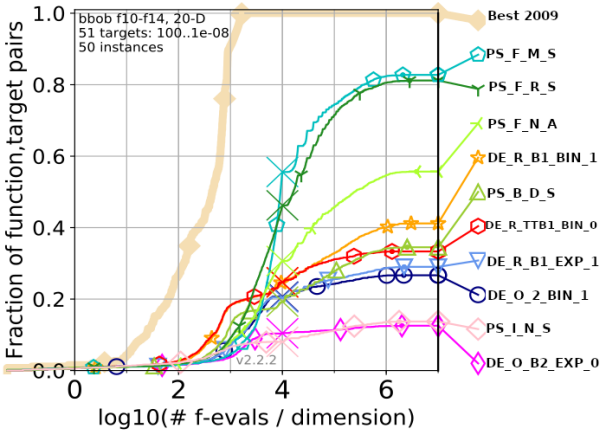
Function group 5, 5-*D*



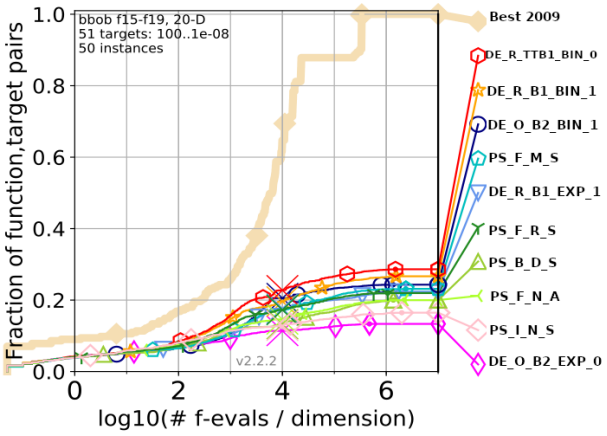
Function group 1, 20-*D*



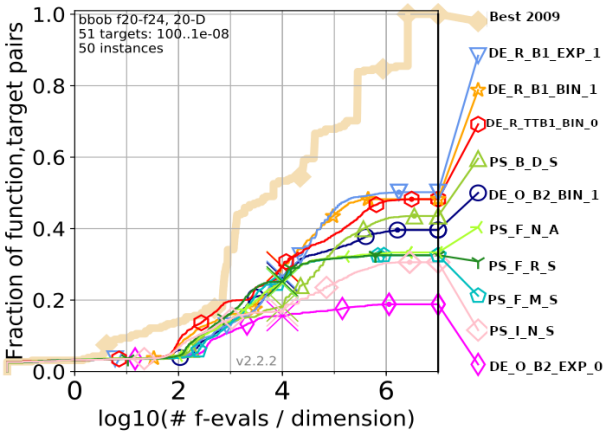
Function group 2, 20-*D*



Function group 3, 20-*D*



Function group 4, 20-*D*



Function group 5, 20-*D*

Appendix B

aRT Tables

Tables B.1 and B.2 on the next two pages contain the aRT's of every PSO- and DE instance on target $f_{opt} + 10^{-7}$ of test functions 1, 6, 10, 15 and 20, in 2-D and 5-D, divided by the respective best aRT measured during BBOB-2009. The aRT's in 20 dimensions are omitted because in most cases, instances never reached the final target. The most common instances and the best value(s) in every column are highlighted in **bold**.

Table B.1: aRT of every PSO instance on target $f_{opt} + 10^{-7}$ of several test functions, in 2-D and 5-D, divided by the respective best aRT measured during BBOB-2009.

Instance	2-D					5-D				
	f-1	f-6	f-10	f-15	f-20	f-1	f-6	f-10	f-15	f-20
PS_B_D_A	57.00	10.00	∞	4.40	**6.70**	134.00	6.70	∞	∞	**1.20**
PS_B_D_S	50.00	9.10	∞	3.60	7.80	123.00	6.40	∞	116.00	1.40
PS_B_G_A	54.00	10.00	∞	5.00	**6.70**	138.00	6.70	∞	57.00	2.50
PS_B_G_S	**49.00**	**8.60**	∞	3.70	7.60	**121.00**	6.10	∞	57.00	1.90
PS_B_I_A	98.00	21.00	∞	4.80	11.00	287.00	8.80	∞	116.00	**1.20**
PS_B_I_S	82.00	18.00	∞	4.70	12.00	259.00	8.30	∞	∞	1.80
PS_B_L_A	100.00	23.00	∞	4.60	11.00	342.00	17.00	∞	∞	3.70
PS_B_L_S	82.00	20.00	∞	6.50	13.00	305.00	16.00	∞	58.00	2.50
PS_B_M_A	87.00	20.00	∞	3.90	9.00	292.00	10.00	∞	38.00	1.40
PS_B_M_S	83.00	18.00	∞	3.80	8.30	271.00	10.00	∞	115.00	1.80
PS_B_N_A	66.00	13.00	∞	4.30	8.90	217.00	9.20	∞	∞	1.50
PS_B_N_S	60.00	11.00	∞	4.30	7.20	192.00	8.30	∞	∞	1.50
PS_B_R_A	84.00	17.00	∞	5.20	8.70	248.00	11.00	∞	116.00	1.40
PS_B_R_S	73.00	14.00	∞	6.00	7.70	229.00	10.00	∞	38.00	2.60
PS_B_W_A	84.00	15.00	∞	7.20	8.90	225.00	10.00	∞	58.00	2.40
PS_B_W_S	69.00	13.00	∞	5.70	7.20	200.00	9.30	∞	∞	2.00
PS_D_D_A	383.00	28.00	44.00	3.00	17.00	678.00	11.00	17.00	∞	∞
PS_D_D_S	373.00	26.00	46.00	3.40	13.00	661.00	13.00	16.00	∞	∞
PS_D_G_A	371.00	27.00	45.00	3.10	17.00	657.00	12.00	16.00	∞	22.00
PS_D_G_S	376.00	27.00	45.00	3.80	15.00	669.00	21.00	19.00	115.00	22.00
PS_D_I_A	433.00	31.00	50.00	3.30	14.00	782.00	11.00	16.00	∞	15.00
PS_D_I_S	440.00	31.00	50.00	3.40	12.00	772.00	11.00	16.00	57.00	15.00
PS_D_L_A	461.00	33.00	56.00	3.80	15.00	1091.00	18.00	23.00	∞	45.00
PS_D_L_S	469.00	33.00	51.00	4.20	14.00	1077.00	16.00	23.00	∞	22.00
PS_D_M_A	365.00	29.00	50.00	3.50	15.00	790.00	12.00	19.00	117.00	8.60
PS_D_M_S	385.00	28.00	48.00	3.60	14.00	779.00	12.00	18.00	115.00	22.00
PS_D_N_A	415.00	29.00	47.00	3.50	15.00	794.00	11.00	18.00	116.00	7.10
PS_D_N_S	405.00	28.00	46.00	3.20	16.00	785.00	11.00	17.00	∞	15.00
PS_D_R_A	436.00	30.00	54.00	4.30	15.00	859.00	13.00	19.00	∞	22.00
PS_D_R_S	435.00	31.00	52.00	4.40	14.00	839.00	12.00	18.00	116.00	22.00
PS_D_W_A	499.00	33.00	55.00	4.00	18.00	883.00	67.00	88.00	∞	∞
PS_D_W_S	478.00	33.00	53.00	4.00	21.00	846.00	62.00	25.00	117.00	∞
PS_F_D_A	128.00	8.80	118.00	3.90	22.00	961.00	∞	∞	∞	∞
PS_F_D_S	115.00	20.00	550.00	3.80	28.00	153.00	∞	∞	115.00	∞
PS_F_G_A	125.00	9.00	53.00	3.60	23.00	1082.00	∞	∞	∞	∞
PS_F_G_S	115.00	10.00	3280.00	4.20	35.00	152.00	∞	∞	∞	∞
PS_F_I_A	220.00	18.00	30.00	2.30	10.00	447.00	9.20	9.10	4.50	∞
PS_F_I_S	221.00	17.00	29.00	2.80	11.00	434.00	6.50	8.70	4.30	∞
PS_F_L_A	219.00	19.00	29.00	2.60	8.50	660.00	13.00	16.00	58.00	11.00
PS_F_L_S	222.00	17.00	28.00	2.50	10.00	608.00	12.00	15.00	18.00	11.00
PS_F_M_A	265.00	21.00	45.00	2.90	11.00	687.00	13.00	20.00	14.00	2.10
PS_F_M_S	266.00	20.00	43.00	2.60	10.00	665.00	12.00	19.00	10.00	2.90
PS_F_N_A	168.00	13.00	18.00	**1.70**	8.50	365.00	7.50	8.30	6.00	∞
PS_F_N_S	165.00	12.00	17.00	2.10	11.00	335.00	6.00	**7.90**	4.70	22.00
PS_F_R_A	185.00	14.00	22.00	2.20	9.40	424.00	7.90	9.40	6.70	7.20
PS_F_R_S	181.00	14.00	21.00	1.90	8.80	397.00	7.30	9.20	**3.20**	5.00
PS_F_W_A	328.00	47.00	128.00	6.10	46.00	1439.00	466.00	∞	∞	∞
PS_F_W_S	312.00	39.00	131.00	10.00	49.00	1569.00	262.00	∞	∞	∞
PS_I_D_A	117.00	9.30	**16.00**	3.00	17.00	233.00	7.90	10.00	∞	44.00
PS_I_D_S	112.00	9.40	**16.00**	3.10	18.00	218.00	23.00	19.00	∞	11.00
PS_I_G_A	115.00	9.40	**16.00**	4.60	9.00	227.00	12.00	10.00	∞	∞
PS_I_G_S	112.00	9.20	17.00	3.60	14.00	219.00	42.00	124.00	115.00	15.00
PS_I_I_A	141.00	12.00	18.00	3.20	12.00	369.00	6.60	8.50	∞	∞
PS_I_I_S	141.00	11.00	17.00	3.40	12.00	343.00	6.30	8.00	117.00	15.00
PS_I_L_A	142.00	11.00	17.00	3.30	8.40	519.00	12.00	15.00	117.00	22.00
PS_I_L_S	134.00	12.00	17.00	3.40	15.00	476.00	11.00	13.00	∞	44.00
PS_I_M_A	136.00	10.00	**16.00**	3.40	8.70	356.00	7.50	10.00	∞	22.00
PS_I_M_S	126.00	10.00	**16.00**	2.50	10.00	340.00	7.00	9.00	57.00	∞
PS_I_N_A	120.00	10.00	**16.00**	2.20	11.00	299.00	6.10	8.50	∞	22.00
PS_I_N_S	120.00	10.00	**16.00**	2.50	14.00	277.00	**5.30**	8.10	∞	11.00
PS_I_R_A	132.00	10.00	**16.00**	3.60	10.00	334.00	7.50	10.00	117.00	∞
PS_I_R_S	129.00	11.00	**16.00**	2.50	11.00	301.00	6.80	9.00	58.00	15.00
PS_I_W_A	146.00	11.00	18.00	4.30	16.00	369.00	107.00	57.00	∞	∞
PS_I_W_S	137.00	12.00	18.00	3.60	19.00	322.00	76.00	25.00	∞	45.00

Table B.2: aRT of every DE instance on target $f_{opt} + 10^{-7}$ of several test functions, in 2-D and 5-D, divided by the respective best aRT measured during BBOB-2009.

Instance	2-D					5-D				
	f-1	f-6	f-10	f-15	f-20	f-1	f-6	f-10	f-15	f-20
DE_O_B1_BIN_0	49.00	**5.10**	22.00	2.80	11.00	114.00	**3.10**	17.00	58.00	2.70
DE_O_B1_BIN_1	54.00	8.00	24.00	1.80	11.00	129.00	3.70	14.00	22.00	2.30
DE_O_B1_EXP_0	50.00	**5.10**	13.00	3.40	11.00	145.00	4.00	12.00	29.00	1.30
DE_O_B1_EXP_1	54.00	7.20	27.00	2.80	13.00	164.00	4.80	11.00	11.00	0.84
DE_O_B2_BIN_0	70.00	7.40	8.10	1.10	7.70	207.00	7.00	13.00	∞	1.20
DE_O_B2_BIN_1	69.00	7.20	7.60	1.00	7.90	198.00	6.10	17.00	29.00	0.88
DE_O_B2_EXP_0	71.00	7.50	8.10	1.20	6.80	245.00	9.00	13.00	∞	0.52
DE_O_B2_EXP_1	70.00	7.10	7.80	**0.97**	9.00	226.00	6.90	17.00	57.00	0.44
DE_O_NS_BIN_0	110.00	12.00	13.00	2.00	4.80	370.00	9.30	37.00	8.40	0.24
DE_O_NS_BIN_1	100.00	14.00	12.00	1.80	4.50	319.00	8.70	41.00	16.00	0.32
DE_O_NS_EXP_0	109.00	11.00	13.00	1.80	4.80	382.00	11.00	46.00	10.00	0.23
DE_O_NS_EXP_1	95.00	15.00	11.00	2.00	6.00	327.00	10.00	37.00	13.00	0.24
DE_O_R1_BIN_0	88.00	72.00	14.00	2.30	4.40	295.00	7.40	20.00	57.00	0.26
DE_O_R1_BIN_1	81.00	1095.00	16.00	3.00	18.00	269.00	30.00	26.00	9.50	0.56
DE_O_R1_EXP_0	88.00	69.00	14.00	2.10	6.50	315.00	8.90	21.00	∞	**0.20**
DE_O_R1_EXP_1	85.00	917.00	12.00	2.80	14.00	278.00	132.00	26.00	11.00	0.25
DE_O_R2D_BIN_0	67.00	7.00	11.00	4.00	5.20	261.00	6.40	22.00	116.00	0.66
DE_O_R2D_BIN_1	71.00	21.00	12.00	5.30	8.90	261.00	6.20	25.00	57.00	1.00
DE_O_R2D_EXP_0	65.00	6.70	21.00	4.60	6.90	274.00	8.30	26.00	57.00	0.28
DE_O_R2D_EXP_1	72.00	12.00	14.00	5.60	8.80	276.00	7.00	28.00	38.00	0.40
DE_O_R2_BIN_0	106.00	11.00	13.00	1.50	4.80	422.00	11.00	∞	∞	∞
DE_O_R2_BIN_1	90.00	100.00	11.00	1.80	5.10	326.00	8.20	∞	116.00	1.90
DE_O_R2_EXP_0	106.00	11.00	13.00	1.60	4.90	420.00	13.00	∞	∞	3.00
DE_O_R2_EXP_1	93.00	117.00	11.00	1.60	8.90	324.00	8.70	∞	∞	0.89
DE_O_TOP_BIN_0	79.00	31.00	29.00	2.80	13.00	164.00	906.00	140.00	∞	4.00
DE_O_TOP_BIN_1	140.00	389.00	27.00	2.30	11.00	172.00	289.00	34.00	116.00	1.40
DE_O_TOP_EXP_0	69.00	29.00	21.00	3.00	13.00	197.00	51.00	97.00	∞	5.90
DE_O_TOP_EXP_1	103.00	426.00	24.00	2.10	14.00	201.00	354.00	25.00	∞	1.30
DE_O_TTB1_BIN_0	68.00	5.60	11.00	1.20	10.00	160.00	5.50	**10.00**	8.20	2.20
DE_O_TTB1_BIN_1	65.00	72.00	13.00	1.30	11.00	162.00	247.00	14.00	6.30	1.60
DE_O_TTB1_EXP_0	66.00	8.50	10.00	1.30	11.00	203.00	4.20	11.00	57.00	1.20
DE_O_TTB1_EXP_1	64.00	59.00	14.00	1.20	7.30	198.00	33.00	13.00	38.00	0.77
DE_R_B1_BIN_0	49.00	7.00	23.00	3.70	11.00	**113.00**	**3.10**	16.00	18.00	4.50
DE_R_B1_BIN_1	52.00	8.50	22.00	2.30	17.00	128.00	3.80	12.00	37.00	1.70
DE_R_B1_EXP_0	**47.00**	5.20	20.00	3.10	13.00	145.00	4.20	**10.00**	16.00	1.70
DE_R_B1_EXP_1	53.00	11.00	23.00	3.70	14.00	160.00	4.80	11.00	16.00	0.77
DE_R_B2_BIN_0	71.00	7.60	8.30	1.00	8.30	205.00	7.10	13.00	∞	0.76
DE_R_B2_BIN_1	68.00	7.50	7.90	1.20	5.80	202.00	5.90	17.00	116.00	1.40
DE_R_B2_EXP_0	72.00	7.60	8.30	1.20	6.60	245.00	9.10	13.00	∞	0.50
DE_R_B2_EXP_1	70.00	7.20	7.70	1.10	6.20	230.00	6.90	17.00	16.00	0.38
DE_R_NS_BIN_0	111.00	12.00	14.00	2.00	4.50	383.00	10.00	51.00	22.00	0.35
DE_R_NS_BIN_1	99.00	16.00	12.00	1.90	6.00	322.00	8.80	43.00	7.90	0.30
DE_R_NS_EXP_0	112.00	12.00	13.00	2.70	4.40	398.00	11.00	42.00	15.00	0.21
DE_R_NS_EXP_1	101.00	14.00	12.00	1.90	5.70	332.00	10.00	50.00	15.00	0.26
DE_R_R1_BIN_0	89.00	89.00	10.00	2.00	4.70	302.00	7.30	21.00	∞	0.25
DE_R_R1_BIN_1	84.00	1055.00	9.30	2.00	11.00	271.00	35.00	26.00	7.20	0.42
DE_R_R1_EXP_0	92.00	93.00	10.00	1.20	**3.20**	317.00	9.00	21.00	∞	0.21
DE_R_R1_EXP_1	84.00	1065.00	11.00	2.70	14.00	282.00	134.00	26.00	7.80	0.38
DE_R_R2D_BIN_0	66.00	7.00	12.00	3.80	6.90	262.00	6.40	24.00	57.00	0.68
DE_R_R2D_BIN_1	73.00	11.00	13.00	5.40	6.20	266.00	6.10	27.00	22.00	0.99
DE_R_R2D_EXP_0	67.00	6.90	**7.50**	4.10	7.20	277.00	8.20	27.00	116.00	0.39
DE_R_R2D_EXP_1	69.00	21.00	8.10	4.60	7.30	284.00	7.10	25.00	∞	0.36
DE_R_R2_BIN_0	110.00	11.00	13.00	1.40	4.80	433.00	11.00	∞	∞	∞
DE_R_R2_BIN_1	92.00	101.00	11.00	1.40	5.70	325.00	8.20	2822.00	∞	2.10
DE_R_R2_EXP_0	110.00	11.00	13.00	1.50	4.70	423.00	13.00	∞	∞	3.20
DE_R_R2_EXP_1	93.00	99.00	11.00	1.60	6.40	329.00	8.70	∞	∞	0.78
DE_R_TOP_BIN_0	109.00	39.00	55.00	4.50	16.00	245.00	7.70	18.00	∞	11.00
DE_R_TOP_BIN_1	258.00	421.00	38.00	3.20	14.00	200.00	6.50	30.00	116.00	1.20
DE_R_TOP_EXP_0	76.00	44.00	43.00	3.80	16.00	279.00	9.10	19.00	∞	45.00
DE_R_TOP_EXP_1	61.00	389.00	37.00	2.80	11.00	199.00	598.00	25.00	∞	1.30
DE_R_TTB1_BIN_0	70.00	9.50	13.00	1.30	12.00	160.00	4.60	12.00	6.40	2.50
DE_R_TTB1_BIN_1	66.00	58.00	14.00	1.40	13.00	161.00	135.00	12.00	**5.80**	1.40
DE_R_TTB1_EXP_0	67.00	8.30	17.00	1.30	8.70	202.00	4.20	**10.00**	∞	0.99
DE_R_TTB1_EXP_1	65.00	91.00	16.00	1.30	11.00	199.00	45.00	13.00	28.00	0.75